Title: Cost and Benefit Elements of a Tailored Application versus an off-the-Shelf Package and the Manager's Role.

Course:
Year: 1993
Author(s): F. Rivera

Report No: P93060

Abstract: The author of the paper performs a cost-benefit analysis to determine whether to make or buy a software package. If a program is developed in house, whatever the organizational Software Department could have accomplished during the time it spent developing the system is determined. However, adoption of an off-the shelf package means that the benefits of having a "perfect" package will be lost. The opportunity cost would be an estimate of those benefits lost over the period the software packaging is to be used, minus the additional expenses involved in developing a "perfect" package.

Cost and Benefit Elements of a Tailored Application vs. an Off-The-Shelf Package and the Manager Role

F. Rivera
P9360

RESEARCH PAPER

EMGT 535

COST AND BENEFIT ELEMENTS OF A TAILORED APPLICATION VERSUS AN OFF-THE-SHELF PACKAGE AND THE MANAGER ROLE

PORTLAND STATE UNIVERSITY

NOVEMBER 30, 1993

FERNANDO E. RIVERA

Introduction

Today, there are three ways of meeting software need: a package can be purchased, software can be developed in house, or a hybrid "customized" off-the-shelf package can be developed. Of the three alternatives, off-the-shelf packages are often the least expensive, the most reliable, and require the lest time to implement. Their performance and the quality of their documentation and training programs can be known before from vendor information and other package users, so that there are less likely to be any significant surprises after implementation. However, a package that suits organizational needs well, but cannot be adequately supported, can be much less useful in the long run that less well designed package that it is well supported.

In house development may be unavoidable if an application is unique or uncommon capabilities or features are required. I has the advantage of being able to meet all requirements, but often requires extensive debugging and requires a long time to implement.

Customization of package software has some of the advantages and disadvantages of each of the others above. It is most appropriated in case where input or output formats of a package program are not appropriated to applications requirements, or additional reports need to be generated from data stored. A rule offered bay many people with software package experience is: if more that 10% - 15% of the package needs to be modified, it is generally more cost-effective to develop the system in house[7].

Commercial software packages are copyrighted and firms often place restrictions on the package they sell or lease, including restrictions on modification. If a package developer do not want to release the source code, internal modifications may be not be possible.[1]

A cost-benefit analysis should be performed to determine whether to make or buy a package. If a program is developed in house, whatever the organizational Software Department could have accomplished during the time it spent developing the system must be given up. However, adoption of a off-the-shelf

---

[1] Sometime hacker or "guru" programmers can made modifications in packages, changing the assembler code directly in the executable module, but it could be considered a copyright violation.

package means that the benefits of having a "perfect" package will be lost. The opportunity cost would be an estimate of those benefits lost over the period the software package is to be used, minus the additional expenses involved in developing a "perfect" package.

Opportunity costs are hard to quantify precisely, but can be among the most important factors in software selections. Wherever a significant benefit can be estimated, it should be done. They may still be some degree of uncertain, but careful thought and analysis can help in the identification of significant opportunity cost.

Chapter 1 Software Development

1.1 Life Cycle

1. Requirement

The first phase of the Software Development Life Cycle, is initiated by a statement of need. This need can be expressed by a user under the form of a 'Requirements Definition' or can be a result of business objectives. A preliminary analysis of the concept can also be carried out to enable a sizing up of the cost of software development.

A detailed analysis of users requirements is carried out by a product team, a Functional Specification is prepared and a Product Plan is set up. This product Plan must include an estimate of the time and cost of the software to be developed.

2. Software Design

The project is in accordance with the Product Plan. The project then proceeds in two steps. Firstly, the system architecture is established: this determines the hardware and software trade-offs. Then, each subsystem is specified to satisfy the Functional Specifications. As a results, the subsystem specifications are produced which allow the Product Plan to be redefined. The software cost and time estimates play an important role here as it can influence the hardware and software trade-offs.

3. Program Construction

During the construction phase, the software modules are coded and tested. The construction costs and time of a system will vary widely, depending upon such factors as the type of applications, the product complexity, language level, personnel attributes, use of software tools, etc.

4. Documentation

Software project by nature generate an enormous quantity and diversity of paperwork, documents, diagrams and tables. The cost associated to the time devoted to this activity might represent a significant percentage of the entire cost of the development.

5. Program Test

The tested modules during construction phase are integrated into subfunctions, which are then tested; subsystem are integrated and tested in their target processors and then the system is integrated and tested.

6. Maintenance

During this phase the product is update and adapted to match changing organizational need. The maintenance costs of a system will vary widely, depending upon such factors as the applications, the complexity of the system, and the need for periodic updates.

1.2 Estimations

Success in software engineering depends primarily on managerial considerations, which are in turn contingent on an accurate and reliable cost-estimations procedure. One of the first challenges lies in finding an estimating method to insure that everything that need to be included in the estimate of software development costs are identified. Among the many software cost-estimation procedures available today, COCOMO[3] stand out by its concern for accuracy and the thoroughness of its procedures.

The software estimating process requires a basic understanding of the fact that software development is not a 'mechanic' process, like that which occurs in building constructions. There, task are concrete, visible, measurable by simple means and of finite quantity. By contrast, software development is a probabilistic process, consisting of a large number of task of undetermined complexity[10].

In a survey[9] was found that the majority of the technical staff estimating software costs use informal analogy and high-level partitioning of requirements, and that no formal procedure exists for incorporating risk and uncertainty. The technical staff is significantly better at estimating effort than size.

To get a reliable software cost estimate, we need to do much more that just put numbers into formulas and accept the results. A seven-step process for software cost estimation is provide by Boehm[3], which show that a software cost estimation activity is a miniproject and should be planned, reviewed and followed up. The seven steps are:

1. Establish Objectives
2. Plan for Required Data and Resources
3. Pin Down Software Requirements
4. Work Out as Much Detail as Feasible
5. Use Several Independents Techniques and Sources
6. Compare and Iterate Estimates
7. Followup

The more detail we provide as input to a software cost estimate, the more accurate our resulting estimate is likely to be. A tentative cost driver attributes list is provided as input to a software cost estimate.

1. Product Attributes
   - Required software reliability
   - Data base size
   - Product complexity
   - Type of application

2. Computer Attributes
   - Execution time constraint
   - Main storage constraint
   - Computer turnaround time
   - Hardware configuration
   - Language level

3. Personnel Attribute
   - Analyst capability
   - Applications experience
   - Programmer capability
   - Programming language experience
   - Personnel continuity
   - Personnel morale
   - Management quality

4. Project Attribute
   - Modern programming practices
   - Use of software tools
   - Required development schedule

5. User Attribute
   - User interface quality
   - Amount of documentation
   - Requirement volatility

Estimating size is already a difficult task in organizations where a software metrics activity exists, but it is made particularly hard when there is no measurement baseline of previous project.

An equation for estimating the number of man-months (MM) required to develop the most common type of software product, in terms of the numbers of thousands of delivered source instructions (KSs) in the software product, according Boehm[3] is:

$$MM = 2.4(KSs)^{1.05}$$

He also presents an equation for estimating the development schedule (DS) in month:

$$DS = 2.5(MM)^{0.38}$$

A metric assistance to size estimating was proposed by Albrecht[1]. He analyzed the statement of requirements of 24 data processing programs in term of the following:

- Inputs
- Outputs
- Inquiries
- Files
- Interfaces

Albrecht then adjusted these numbers according to three levels of complexity -simple, average and complex - to within a range of -25% - +25%. Next, he carried out a statistical analysis on these numbers in relation to the size of the programs and the effort they required for development. During the course of this work, he discovered a high degree of correlation between the size of data processing type of programs his company (IBM) and a measure called the Function Points. This measure is defined as the linear combination of the five adjusted terms such that:

$$Fp = a.inputs + b.outputs + c.inquires + d.files = e.interfaces$$

where a,b,c,d and e are constant as determined by his statistics.

Albrecht also determined a series of simple equations, using the Function Points, that enable the calculation of the estimated size and cost for some languages used in his organization. As an example, here is the formula used to estimate the size of a Cobol program when its Function Points is known:

$$S = 118.7Fp - 6490$$

In practice, once a software requirements is known, a top-level analysis can provide the Function Points count. Then, by using the appropriate formula - the size formula in this case

- the size estimate can be obtained.

To convert the estimate of size S into estimate of time, effort and cost, organization must look at past project.

Another metric assistance to size estimating was proposed by Putnam[15]. For each module, the size is given by a set of three values: So (optimistic size), Sm (most likely size) and Sp (pessimistic size). From these values, the expected module size, Smd, can be deduced by the formula:

$$Smd = (So + 4Sm + Sp)/6$$

To convert the estimate of size into estimate of time, effort and cost, Putnam considers that there is a fundamental relationship in software development between the number of source statements in the system and the effort, development time and the state of the technology being applied to the project. The equation that describes this relationship is:

$$Ss = Ck \; K^{1/3} \; t^{4/3}, \text{ where}$$

Ss is the numbers of end product source lines of code delivered
K is the life cycle effort in man-years
t is the development time
Ck is a state of technology constant

Ck is determined by the use of modern programming practices, the language used and the development environment among other factors. While Ck is difficult to determine from its individual components, this value can be calibrated for an organization by looking at past project.

Chapter 2 Software Package

2.1 Selection Process

Before intensive analysis and comparison of candidate packages can begin, the number of packages should be narrowed down to manageable number: no more that three to five candidates[7], which will then be closely analyzed. The first step is to verify that each package fulfills the essentials requirements.

An straightforward nine steps approach is presented to help in the package selection process.

1. Requirement
   - Formation of the requirements analysis team
   - Definitions of current procedures
   - Identifications of restriction
   - Estimations of package life

2. Documentation
   - Development of functional specifications
   - Documentation of requirements

3. Identification of candidate

4. Assessment of support needs
   - Documentation review
   - Modification support review
   - Installation support review
   - Training support review
   - Maintenance support review

5. Selection
   - Solicitation of proposals
   - Proposal evaluation
   - Package quality evaluation
   - Vendor evaluation
   - Support evaluation
   - Cost-benefit analysis
   - Final package selection

6. Contract negotiation
   - Package contract negotiation
   - Support services contract negotiation
   - Specifications of performance guarantees
   - Determination of compensation arrangements

7. Installation
   - Formation of the installation team
   - Identification of installation matters
   - Package customization
   - Employee training

8. Testing
   - Identification of test goals

9. Acceptance
   - Identification of acceptance parameters

2.2 Cost Accounting

In order to determine whether the purchase of a software package is the most cost-effective option, the following cost should be taken into consideration:

1. Negotiation

Negotiation cost are those expenses incurred by an organization through expenditure of employee time during the package selection process (see 2.1).

2. Purchasing

If a package is bought or leased, the purchasing cost is simply the price of the software, documentation, training, and modifications to the program. Vendors may offer all these services together or separate documentation and training from the software and sell services separately. If multiple copies of the package are needed, the vendor may offer site licenses as an alternative.

3. Implementation

Implementation cost are those expenses incurred when installing the new software. They include the time an organization's computer must be inoperative while the new software is being installed, the cost associated with testing and parallel operations until the reliability of the new system is assured and training employees to use the system.

Another major implementation cost if data conversion, which involves incorporating old record into

the new system's format. If current data if not computerized, this requires manually entering records into the new system. If current data are already computerized, they will have to be converted into the new system's record format. This usually requires development of a special software, or program, to convert old files to the new ones.

4. Training

The training cost is the cost associated to the time it takes to train employees to use a new package.

5. Software errors

This are the costs of errors discovered after the initial integration of a system. Software errors discovered after integration are usually expensive[14]. The principal cost may not be correcting the programming error, but rather undoing the damage that the error caused. For example, a software error may result in a large number of records containing incorrect information. The impact of this errors can be quite significant.

If software is purchased, the maintenance provider will often correct software errors for a specific period of warranty without additional charges.

6. Maintenance

These are the costs to update and adapt software to match changing organizational need. As an example, a payroll program might have to be modified to reflect a change in tax rates. The maintenance costs of a system will vary widely, depending upon such factors as the applications, the complexity of the system, and the need for periodic updates. Maintenance costs for packaged software might be included in a maintenance contract.

Chapter 3 Software Acceptance and Management

3.1 Software Acceptance Plan

Software acceptance is a process of approving or rejecting software system during development, maintenance or purchase, according to how well the software satisfies pre-defined criteria. The final acceptance decision occurs with verification that the delivered documentation is adequate and consistent with the executable system and that the complete software system meet all buyer or user requirements. This decision is usually based on software acceptance testing [16]. It consists of tests to determine whether the developed or package system meets predetermined functionally, performance, quality, and interface criteria.

Software acceptance is specified in a formal plan. The software acceptance plan identifies products for acceptance, the specific acceptance criteria[2], acceptance reviews, and acceptance testing.

Examples of information which should be included in a software acceptance plan are:

- Project Descriptions : Type of system, major task system must satisfy; external interfaces; expected norman usage; standards.
- Management Responsibilities : Responsibilities for acceptance activities; resources and schedule requirements; standards.
- Administrative Procedures : Anomaly reports; record keeping; communications.
- Acceptance Testing : Test plan and acceptance criteria; test cases and procedures; test results and

---

[2] According to the Webster's New World Dictionary, a criteria is a standard, rule, or test by which something can be judged.

analyses.

3.2 Manager Responsibility

Managers responsible for software acceptance must ensure that the results of software acceptance activities demonstrate whether contractual requirements meet buyer needs, and whether the delivered software system meets the contractual requirements.

Software acceptance managers apply elements of traditional management (e.g., planning, organizing, controlling, monitoring, providing support, performing cost-benefit and risk analyses) to managing the contractual process of develop or acquiring software. managers must use their technical knowledge of the proposed software system, of risk associated with its development and maintenance, and of its expected use to establish the criteria for acceptance.

Conclusions

If no package can be found that satisfy the essential requirements, custom development may be necessary. If a package will be acceptable only after some degree of customizations, the cost of any required package modifications should be determined. It a package is sold with out source code, tailoring may be impossible.

On the other hand, if one or more software packages have been found that are suitable for the applications and conform closely to requirements, and off-the-shelf package is appropriate.

Table 1 provides a synopsis of the relative costs of packaged versus developed software and Appendix I a list of cost-benefit impact elements.

Software acceptance is a contractual process with buyers or users and vendors or developers, respectively, identifying products and criteria for the acceptance of software systems.