An Application of Decision Theory for the Evaluation of Software Prototypes

Sergio R. Cárdenas, Jianhui Tian, and Marvin V. Zelkowitz

Institute for Advanced Computer Studies and Department of Computer Science, University of Maryland, College Park, Maryland

There is a need to quantify management decision making in software development. To this end, this article presents an application of concepts from economic decision theory to an aspect of this problem currently under study by many in the software engineering community — how to prototype a design and evaluate the effectiveness of this prototype. The role of prototyping as an experimental tool is analyzed and techniques for defining what to prototype and when it is cost effective are introduced. We demonstrate some of these decision theory techniques to define a model that management can use to select solutions from a set of alternative designs.

INTRODUCTION

Good management is often characterized by the ability to make correct decisions from incomplete information. A good manager must determine the most probable future events from whatever information is currently available. However, we can never know the future with certainty, so there is always a degree of risk in whatever course of action is undertaken.

Software management is no different. A manager must determine how best to allocate available resources (e.g., people, equipment), develop schedules and milestones, and make software design decisions that will best fulfill some future goals for a project.

Currently there is little in the way of a theory of software management decision making. Most papers on the subject give rules of thumb, ad hoc experiences, and general design methods, with a touch of mythology and the admonition that if you follow such directions, you will generally succeed. What is needed is a more scientific basis for decisions consisting of scientific principles, formulas, and algorithms that can guide the manager in making such decisions.

Development of such principles has two beneficial effects:

- For good managers, the quantification of their decision methods justifies their choices and allows them to discover new relationships that they might not have seen.
- Alas, not every manager is "good." Quantification of these methods permits these managers to make more reasonable decisions and removes some of the randomness in the process.

In reflecting on this discussion of decision making for software, terms such as "probable," "certainty," and "risk" appear. These are all terms that have been studied within the context of economic decision theory, and the relationship of such theory to software management has already been identified [1, 2]. We have studied such concepts and believe that they are applicable to many of the current problems of software management. In particular, we have studied the question, "When is it cost effective to develop a software prototype?" We believe that such decision theory concepts can be applied to software management. This article provides an example of using these techniques for evaluating alternative designs—in particular, in the evaluation of prototypes.

How Does This Relate to Software Management?

Software engineering activities are characterized by a continuous need for management to decide among several requirement constraints, design options, development strategies, methodologies, and tools. Usually there is not enough information to guarantee that the chosen option is the best. Most decisions

Address correspondence to Prof. Marvin B. Zelkowitz, Computer Science Dept., University of Maryland, College Park, MD 20742.

© Elsevier Science Publishing Co., Inc.
655 Avenue of the Americas, New York, NY 10010

0164-1212/92/$5.00

are made under explicit or implicit deterministic assumptions about the client, users and environment, the consequences of these decisions or actions, and future development. These assumptions often stray from reality and the decisions made may not reflect the best that can be done in these situations.

A simplistic solution to the problem might be to optimize, instead, the expected value of the consequence of any decision. This approach can work under specific circumstances where the risk involved is not great. However, most major software management decisions involve great risk; therefore, an approach based on expected payoffs is not adequate.

Consider, for example, the following two choices. The first is to get a guaranteed profit of $1 million. The second is to have a 50% chance of getting either a $5 million profit or a $1 million loss. Data show that about two thirds of all managers would choose the former with an expected guaranteed payoff of $1 million, although the expected value of the riskier latter choice is $2 million [3]. This kind of risk aversion behavior is all too common and important in software engineering to be ignored.

In Section 1 we define the software engineering application to which we wish to apply this model—the need to evaluate the effectiveness of prototyping a software design as part of a software requirements analysis activity. In Section 2 we summarize the relevant aspects of decision theory, and in Section 3 we apply this theory to the problem of software prototypes. The major issues we deal with include:

- how to perform risk analysis in making decisions;
- whether, when, and how to use prototypes to extract information;
- an algorithmic description of the decision process.

As a final introductory comment, realize that we view this process as providing additional information to the software manager. Our technique is based on estimations and preferences of the software manager and results from prototyping experiments. This process provides additional data points to the decision maker who must ultimately make such decisions based on these and other available data.

1. PROBLEM DEFINITION

For decision making under uncertainty, we need (*1*) a statement of both objective and subjective preferences, and (*2*) a process for obtaining more information. In general, perfect information is impossible or too expensive to get. As a result, some degree of subjective judgement and preference is usually unavoidable.

In software engineering, to extract more information, we often need to prototype. Usually there are several alternatives to choose for a software project. Each alternative is associated with different outcomes depending on which operational environment is ultimately true.

The relevant aspects of the state of the world are those related with "software needs." Any software product should be built in accordance with the state of nature in which it is going to work. For example, in what domain will the program operate? Are big or small data files appropriate? Are executions long or short? What is the maximum number of transactions per second? Depending on which state of nature holds, the same set of software solutions can be ranked in different ways, since some alternatives may be more profitable than others for a given state and some can even be unfeasible or unacceptable for other states. When we are interested in a future state of the world, it is obvious that we cannot know with absolute certainty which state will hold.

1.1 Prototypes

Building a prototype gives a designer information useful for larger projects. But we must bear in mind that extracting this information is not free. We must weigh the gain versus cost of this new information.

The choice of whether to make immediate decisions or to prototype and then decide depends on the value of information gained from prototyping versus the cost of prototyping. If the value gained is greater, we will prototype and make our decision later; otherwise an immediate decision should be made. Naturally, this rule can be used iteratively at each stage of decision making. It may be necessary to prototype more than once to reach a final decision. The termination conditions are that either further prototyping costs more than gained value of the additional information, or we have already acquired (near-)perfect information. Note that this, in essence, implements a spiral model risk-reduction life cycle to software development [4].

The likelihood of each state of nature is the major uncertainty we consider here. To get a better estimate, experimentation in the form of prototyping may be necessary. Several authors consider prototyping as a technique for providing a reduced functionality version of a software system early in its development [5–8]. We basically agree with this definition but consider that prototypes should be used not only

to experiment with functionality, but also to experiment with any desired attribute of the system. We also consider that prototypes can be used as a source of information throughout the whole life cycle process (e.g., maintenance changes can be first simulated using a prototype).

"To appreciate the rapid prototyping paradigm, you must view software design as an iterative decision-making process" [9]. We view software prototypes as programs that model some aspect of the final product, but often fail to model others. Following the model of (basic) requirements as a vector of attributes [10], we see the requirements of the prototypes as a subset of the requirements (a projection of the vector) of the desired program. Then, from the program requirements, it is possible to build several different prototypes, each satisfying different aspects (each aspect is specified by a subset of the requirements) of the program.

We characterize prototypes as providing answers to the following three questions:

Do requirements reflect client's needs? In this case, the experiment involves interaction between the prototype and the client. This interaction will tell us if an aspect of the client's needs is captured by the corresponding aspect of the requirements modeled by the prototype. This kind of interaction arises very often in software projects: "... it is really impossible for a client, even working with a software engineer, to specify completely, precisely, and correctly the exact requirements of a modern software product before trying some versions of the product" [11]. A prototype will give information of which state of nature holds: client likes aspect x of requirements r, or client does not like aspect x and prefers x'. User evaluations can then be incorporated as feedback to refine the emerging system specifications and designs.

Do requirements reflect system environment needs? In this case, the prototype is built to interact with the environment in which the final system is intended to work. Usually the interaction is simulated; that is, prototyping includes the construction of a model of the environment, and the interactions are only between the prototype and the model. The different states of nature define the degree to which the interactions between environment and system are satisfactory.

Are the requirements feasible? This situation requires an experiment to show if the resources available to the software developers are enough to build a product with the specified requirements. The prototype should model the aspect of the requirements suspected to be unfeasible. In this case, feasibility information is obtained directly from the possibility to construct the prototype within some scaled resource limits. A prototype will give information of which state of nature holds: the aspect x of requirements r is not feasible, or the aspect x is feasible.

A software tool to build prototypes, or a system for computer-aided rapid prototyping (fast prototyping), allows developers rapidly to construct concrete executable models of selected aspects of a proposed system. We argue that such a tool must either predict or bound the time and effort needed to build a given prototype in order to extract the information needed.

First, we consider the decision rules when no further information gathering is allowed. A decision taken in such situations is called an immediate decision. For immediate decisions we analyze the case of complete uncertainty and the case of a known probability distribution. Later we consider the general case in which a delayed decision can be reached after new information is obtained, such as with a prototype.

2. ASPECTS OF DECISION THEORY

The following briefly summarizes the basic model of decision theory [12, 13] that applies to our software prototyping problem.

Assuming several possible strategies (e.g., prototyping experiments) with specified probabilities of a set of possible outcomes (e.g., the possible results), how do we choose a best strategy (e.g., which prototyping experiment provides more information)?

Consider two solutions for a software requirement named alternative designs A_1 and A_2.

A_1: its cost of development is estimated to be 50 monetary units. It is cheap compared with A_2 but it involves more risk.

A_2: It costs 100 monetary units. It represents a conservative but more expensive option.

Assume there are three possibilities for states of nature, labeled S_1, S_2, and S_3, respectively.

S_1: This state is more favorable for alternative A_2 than for alternative A_1. The revenue related to alternative A_1 is 150 and to A_2 is 400.

S_2: This state is more favorable for alternative A_1 than alternative A_2. A_1 gives a revenue of 550 and A_2 a revenue of 300.

S_3: This state is unfavorable for alternative A_1, with

a revenue of 50 monetary units, but alternative A_2 gives a revenue of 300.

We can first identify the payoffs for each alternative under different states of nature. Here the payoffs were calculated by subtracting the cost of development of that alternative from the revenue related to the alternative for the state. This information is summarized in the monetary payoff matrix Y (all entries are expressed in monetary units):

$$Y = \begin{bmatrix} 150 - 50 & 550 - 50 & 50 - 50 \\ 400 - 100 & 300 - 100 & 300 - 100 \end{bmatrix}$$

$$= \begin{bmatrix} 100 & 500 & 0 \\ 300 & 200 & 200 \end{bmatrix} \quad (1)$$

Entry $y_{i,j}$ in the matrix gives the payoff of the alternative A_i (ith row) when S_j is the resulting state of nature. For example, if we choose A_1 and the state of nature turns out to be S_2, we get a payoff of \$500, or equivalently, $y_{1,2} = 500$. A particular combination of alternative and state of nature is called an outcome.

2.1 Immediate Decision Under Complete Uncertainty

Often, when the probability for each state of nature is not known, there are two approaches for decision making without further information. One is to make the decision based on a guaranteed lower or upper bound. The other is to make the decision assuming that all states have the same probability of occurrence. The former approach is often chosen because the latter is based on a dubious assumption.

We call this the maximin rule. The alternative picked by this rule is the one that has the greatest guaranteed minimal payoff. This guaranteed payoff, called maximin value Ω, can be calculated as follows:

$$\Omega = \max_i \left(\min_j y_{i,j} \right) \quad (2)$$

In our example, we choose alternative A_2 because its minimal payoff is 200, which is greater than the minimal payoff of 0 for alternative A_1.

2.2 Immediate Decision under Known Probability Distribution

In some situations, probabilities are known or can be estimated for the different states of nature. Given the probability distribution vector P, where p_i is the probability that state of nature S_i is true, we must decide which alternative to choose.

When the difference between the smaller and the larger payoff for each alternative is relatively small, a decision rule based on expected payoffs may be adequate. We denote the expected payoff for alternative A_i as v_i:

$$v_i = \sum_j y_{i,j} p_j \quad (3)$$

The expected payoff decision rule is: choose A_i, which maximizes v_i, or:

$$\max_i \left(\sum_j y_{i,j} p_j \right) \quad (4)$$

or

$$\max_i (v_i)$$

For example, if we know that the probability distribution for each state of nature in our example is $P = (0.3, 0.5, 0.2)$, we can calculate the expected payoffs as follows:

$$v_1 = 100 \times 0.3 + 500 \times 0.5 + 0 \times 0.2$$
$$= 280$$
$$v_2 = 300 \times 0.3 + 200 \times 0.5 + 200 \times 0.2$$
$$= 230$$

We would then choose A_1 over A_2 since $280 > 230$.

When the stakes involved are enormous (e.g., there are significant differences between the smallest and largest payoffs for some of the alternatives), the expected payoffs decision rule often is inadequate. For example, if the payoffs in matrix Y represent millions of dollars, alternative A_2 would probably be chosen with its guaranteed minimal profit of \$200 million in states S_2 and S_3, although alternative A_1 gives a higher expected value but has a potential of \$0 profit if state S_3 should turn out to be true.

2.3 Risk Analysis

Risk aversion plays an important, even dominan, role in decision making. The technique described below depends on the following assumption about reasonable behavior employed in decision making under uncertainty:

- Decomposition: given three payoffs $y_1 \le y_2 \le y_3$, there exists a probability ρ such that the decision maker is indifferent to the choice of a guarantee of y_2, and the choice of getting y_3 with probability ρ and getting y_1 with probability $1 - \rho$.

For example, say there are two techniques to solve a problem, one that is fully tested giving a guaran-

teed payoff of \$5,000 and a second new and more efficient (but not completely tested) technique promising a potentially larger payoff of \$10,000 but with a chance to give a payoff of only \$2,000. Some software managers will consider using the new technique only if the chances of getting the payoff of \$10,000 are $> 80\%$. In this case, probability ρ is approximately 0.8.[1]

2.4 Decomposition of $y_{i,j}$

Let y_0 be the minimal value in our payoff Y and let y^* be the maximal value. In our example (section 2, equation 1), we would choose $y^* = 500$ and $y_0 = 0$. With the selection of y_0 and y^*, we can decompose each $y_{i,j}$ of Y as follows based on the decomposition assumption given earlier:

- What is the equilibrium probability $e_{i,j}$ that makes you indifferent to the two choices $y_{i,j}$, or probability $e_{i,j}$ of getting y^* and probability $1 - e_{i,j}$ of getting y_0?

The answer to this question is a subjective determination that depends on current as well as past experiences, and reflects the risk aversion of the decision maker. Note that if $y_{i,j} = y_{k,l}$, then $e_{i,j} = e_{k,l}$, so redundant entries need only be evaluated once. Any element $e_{i,j}$ will satisfy the following inequality:

$$y_0 \times (1 - e_{i,j}) + y^* \times e_{i,j} \geq y_{i,j} \tag{5}$$

The difference between the two sides of equation 5 reflects the degree of risk aversion. If the two sides are equal, risk analysis reduces to the expected value approach. That is, when the expected value is the same, the two outcomes are equally desirable. Also, we do not model risk seeking as in lotteries, where one's expected monetary return (probability to win $\times$ lottery price) is less than the price one pays (e.g., the left side of the equation is less than the right side). However, there is nothing intrinsic in our approach to prevent one from modeling such behavior.

Depending on $e_{i,j}$, we decompose the payoff $y_{i,j}$ into an equivalent pair of payoffs $\{y_0, y^*\}$, with probability $e_{i,j}$ of getting the more desirable y^*. We call the matrix formed by these elements $e_{i,j}$ the equilibrium matrix E. The probability $e_{i,j}$ captures the desirability of payoff $y_{i,j}$ in terms of boundary payoffs.

Following our example (section 2, equation 1), we need to ask the above question for the payoffs \$100, \$200, and \$300. For example, we might reply with the following possible result based on our own risk aversion behavior:

$$E = \begin{bmatrix} 0.3 & 1 & 0 \\ 0.8 & 0.6 & 0.6 \end{bmatrix}$$

2.5 Decomposition and Comparison of Alternatives

We next evaluate the desirability of different alternatives. Essentially they are also a decomposition in terms of the boundary payoffs. The desirability for an alternative A_i, denoted as d_i, can be calculated as follows:

$$d_i = \sum_j p_j \times e_{i,j} \tag{6}$$

The comparison of different alternatives boils down to the comparison of their desirabilities. The rule can be simply stated as:

$$\max_i d_i \tag{7}$$

The alternative A_i with the maximal desirability d_i will be selected. In our example, we have:

$$d_1 = 0.3 \times 0.3 + 1 \times 0.5 + 0 \times 0.2 \approx 0.6$$

$$d_2 = 0.8 \times 0.3 + 0.6 \times 0.5 + 0.6 \times 0.2 \approx 0.7$$

The values d_1 and d_2 can be interpreted as decomposition probabilities. Instead of having different values of desirability (decomposition probabilities) for each alternative, one for each state of nature, we derive the expected desirabilities d_1 and d_2. We choose A_2 because d_2 is larger than d_1.

2.6 Value and Usage of Extracted Information

When faced with decision making under uncertainty, we may choose to get more information so that a better final decision can be made. If this option is available, the decision can be delayed after the extraction of information, allowing the decision to be based on a more accurate information base. However, before undertaking the procedure to extract more information, we have to make sure that the gain achieved by more information will outweigh the cost of obtaining it. Here, we try to establish an absolute boundary: what is the value of perfect information?

[1] Utility functions are used to address the same problem. A utility function maps payoffs to numbers (not necessarily probabilities). The numbers give the degree of desirability of the payoffs. Here we do not define a function from payoffs to numbers; instead, we obtain the decomposition probability only for the payoffs in matrix Y.

The best we can expect from experimentation is that the experiment results will indicate for sure which state of nature will hold. Under this case, we can choose the alternative that gives the highest payoff under the given state of nature. However, this information cannot be obtained a priori to the experiment. Thus, we have the expected payoff of perfect information Φ:

$$\Phi = \sum_j p_j \times \max_i y_{i,j} \tag{8}$$

In our example, we would choose A_1 under S_2 and choose A_2 otherwise, resulting in optimal payoff Φ:

$$\Phi = 0.3 \times 300 + 0.5 \times 500 + 0.2 \times 200 = 380$$

What is the value of this perfect information? This depends on what we were committed to originally. For example, if we were committed to the maximin rule with a payoff of 200, the value of this information is 380 − 200 = 180; if the expected payoff rule (of 280) was used, we value perfect information at 380 − 280 = 100. The value of perfect information is the maximum we could spend to obtain the information using a prototype.

3. APPLICATION TO SOFTWARE PROTOTYPES

Let us now consider the issue of applying the previous economic model to the problem of evaluating the effectiveness of a software prototype. Assume we experiment (prototype) to determine which state of nature holds.

An ideal experiment would be like a "state-meter" that indicates with perfect accuracy which state is really true. On the other hand, the worst experiment is one having results that are independent of the state of nature. Real experiments (prototyping) fall between these extremes. The results from prototyping depend on the state of nature, but the dependency is probabilistic. Let $result_1, result_2, \ldots result_k$ be the possible results of the prototyping experiment. Each dependency between state and result is expressed as the conditional probability that the result will be $result_i$ given that the state of nature is S_j.

This information will be presented in a conditional probability matrix C, with a row for each different result of the prototype and a column for each state of nature. Each entry $c_{i,j}$ represents the conditional probability of prototyping result $result_i$ given that state of nature S_j holds.

Since we have the probabilities for each state (vector P) and the conditional probability matrix C, it is possible to calculate the probability for each result $result_i$ of the prototype. The probability distribution for prototyping results is summarized in vector Q (marginal probability distribution), where q_i represents the probability of getting $result_i$.

$$q_i = \sum_j c_{i,j} \times p_j \tag{9}$$

Taking into consideration the results of the prototype, it is possible to obtain better estimates for the probabilities of each state (update vector P). Using Bayes rule, we get the a posteriori distribution matrix P', having elements calculated as follows:

$$p'_{i,j} = \frac{c_{i,j} \times p_j}{q_i} \tag{10}$$

Note that while P is a vector, P' is a matrix. P' has as many rows as results from the prototype. Each row is an updated version of vector P. Row i gives the probabilities of the states of nature given that the result of the prototype is $result_i$.

Following our example, assume that a prototype of alternative A_2 is planned. The planned prototype can give the following results:

$result_1$: client is satisfied with the system as presented by prototype.

$result_2$: client is not satisfied.

We first estimate the probabilities $c_{i,j}$ that the results of prototyping are consistent with the true states of nature before the prototyping experiment is performed. For example, we estimate that if the state of nature is S_2 (favorable for alternative A_1 and unfavorable for alternative A_2) we have probabilities 0.3 and 0.7 to obtain results $result_1$ and $result_2$, respectively, from the prototype. The conditional probability conditioned on states of nature is given below:

$$C = \begin{bmatrix} 0.9 & 0.3 & 0.4 \\ 0.1 & 0.7 & 0.6 \end{bmatrix}$$

From this we can calculate the probability q_i for each result i of the prototype $Q = (0.5, 0.5)$ and the a posteriori distribution matrix as:

$$P' = \begin{bmatrix} 0.54 & 0.30 & 0.16 \\ 0.06 & 0.70 & 0.24 \end{bmatrix}$$

For example, if we get $result_1$ from the prototyping study, the new expected value and desirability for the alternatives A_1 and A_2 are:

$$v_1 = 100 \times 0.54 + 500 \times 0.3 + 0 \times 0.16 \approx 200$$

$$v_2 = 300 \times 0.54 + 200 \times 0.3 + 200 \times 0.16 \approx 250$$

$$d_1 = 0.3 \times 0.54 + 1 \times 0.3 + 0 \times 0.2 \approx 0.46$$

$$d_2 = 0.8 \times 0.54 + 0.6 \times 0.3 + 0.6 \times 0.16 \approx 0.71$$

In this case, alternative A_2 should be chosen, since it gives both the higher expected payoff and higher degree of desirability.

Similarly, if we should get $result_2$ from the prototyping study, the new expected value and desirability for the alternatives A_1 and A_2 are:

$$v_1 = 100 \times 0.06 + 500 \times 0.7 + 0 \times 0.16$$
$$\approx 360$$
$$v_2 = 300 \times 0.06 + 200 \times 0.7 + 200 \times 0.24$$
$$\approx 210$$
$$d_1 = 0.3 \times 0.06 + 1 \times 0.7 + 0 \times 0.2$$
$$\approx 0.72$$
$$d_2 = 0.8 \times 0.06 + 0.6 \times 0.7 + 0.6 \times 0.24$$
$$\approx 0.61$$

In this case, alternative A_1 is the obvious choice.

Thus the expected value and expected desirability gains of prototyping study are:

$$v_P = 0.5 \times 360 + 0.5 \times 250 - 230$$
$$\approx 75$$
$$d_P = 0.5 \times 0.71 + 0.5 \times 0.72 - 0.7$$
$$\approx 0.02$$

If the cost of performing this prototyping study is less than $v_P = 75$, this study should be carried out. Otherwise, an immediate decision should be made (see sections 2.1 and 2.5).

3.1 Other Decision Rules

Choice under uncertainty is currently a field in flux. Economists, decision theorists, and psychologists are trying to develop better models for this important area. Several decision rules (usually very complex) have been defined; see reference [14] for an account of problems and proposed solutions.

We do not define an automated process to mimic the behavior of a human being when faced with a decision. Our approach is more prescriptive (we suggest a software alternative in terms of some criteria) than descriptive (we do not claim to suggest the same alternative that a given person will pick).

Our goal is to provide support to the decision maker. The support comes in the form of information: the expected payoff for each alternative; the expected utility for each alternative; the probability for each state of nature and the ranking of the alternatives defined for each state of nature; what to prototype to reduce uncertainty.

3.2 Decision Process Algorithm

To summarize, we state our design approach via the following steps:

- Identify alternatives A_i.
- Identify states of nature S_j.
- Evaluate the monetary payoff for each alternative under each state of nature. Build the matrix $Y = [y_{i,j}]$.
- Use the decision process as a tool to select an alternative (Figure 1.)

There are three major issues to be resolved in deriving the payoff matrix Y: (*1*) we must identify all possible alternatives; (*2*) we must identify all possible states of nature; and (*3*) we must evaluate the payoffs of choosing each alternative under each state of nature.

A first but crude approximation for defining the states of nature is to consider for each alternative that the world will be in only two possible states—it will be favorable or unfavorable for that alternative. Then if we have alternatives A_1 and A_2, then we can define four states of nature: S_1 favorable for both A_1 and A_2, S_2 favorable for A_1 but unfavorable for A_2, etc.

When each alternative defines a different software solution, it is convenient to consider some basic predicates that define the states of nature. We have already classified prototypes as falling into three categories: client needs, system needs, and feasibility. Then, for each solution i we will assume the predicates:

cli_i: is the client willing to pay for the software product?

env_i: are the interactions between system and environment satisfactory?

fea_i: is the software concept feasible with the available resources (people, time, tools, etc.)?

A state of nature is defined as one of the possible combinations for the values of all the predicates. A methodology to identify the relevant states of nature can be summarized as follows:

1. Identify which types of predicates are relevant (e.g., if feasibility is assured, then only predicates for client and environment are considered).
2. Construct the basic set of states of nature in which each state is a different combination of the values of the predicates.

Input:

M : Payoff matrix of alternatives × states of nature

Output:

win : Natural, represents the index of winning alternative

Constants:

$alte$ = number of alternatives, number of rows in Y
$stat$ = number of states of nature, number of columns in Y
$\Omega = \max_i(\min_j y_{i,j})$
$y_0 = \min_{i,j} y_{i,j}$
$y^* = \max_{i,j} y_{i,j}$
mpc = minimal prototype cost

Variables:

$risk$: Boolean, true if risk aversion is significant
$resu$: Natural, number of results for current prototype
pr : Natural, result of prototype experiment
acc : Natural, accumulated cost from prototyping
pay : value expected from alternative
x : Prototype definition
C : Matrix $resu \times stat$ of probabilities
E : Matrix $alte \times stat$ of probabilities
Q : Vector $resu$ elements, of probabilities
P : Vector $stat$ elements, of probabilities
P' : Matrix $resu \times stat$ of probabilities
sav : maximum savings due to the use of prototype
Φ : optimal payoff

Functions:

$Proba(x : payoff)$ = Decomposition probability of x in terms of y_0 and y^*
$Experiment(x : prototype)$ = Result of the prototype x (index of result)
$Defineproto$ = Prototype to help to determine state of nature
$Cost(x : prototype)$ = Cost of prototype x
$Numres(x : prototype)$ = No. of different results of prototype x

Figure 1. Decision process (Data).

```
BEGIN
/* If no further information is affordable since any experiment will be more expensive than
the information obtained, use maximin for an immediate decision. */
if y* − Ω < mpc then win ← alternative i, where ∀j(y_{i,j} ≥ Ω);
else
    P ← [p_i] where p_i is first estimation for probability of state i;
    mid ← (y* − y0)/2;
    /* Check if the difference between the largest and the smallest payoffs is significant
    enough to make risk aversion important */
    if Proba(mid) ≈ 0.5 then
        risk ← false;
        win ← i, where max_i(Σ_j y_{i,j} × p_j);
        pay ← Σ_j y_{win,j} × p_j;
    else
        risk ← true;
        E ← [e_{i,j}], where e_{i,j} = Proba(y_{i,j});
        D ← [d_i], where d_i = Σ_j p_j × e_{i,j};
        win ← j where d_j = max d_i;
        pay ← Σ_j y_{win,j} × p_j;
    x ← Defineproto;
    acc ← Cost(x);
    Φ ← Σ_j p_j × max_i y_{i,j};
    sav ← Φ − pay − acc;
    /* Cost/benefit analysis: While the cost of prototyping is less than the value of the
    extracted information, obtain new information defining and using prototypes */
    while sav > 0 do
        resu ← Numres(x);
        C ← [c_{i,j}], where c_{i,j} cond. prob. of result i given state j;
        Q ← [q_i], where q_i = Σ_j c_{i,j} × p_j;
        P' ← [p'_{i,j}], where p'_{i,j} = c_{i,j} × p_j/q_i;
        pr ← Experiment(x);
        P ← [p_i] where p_i = p'_{pr,i};
        x ← Defineproto;
        acc ← acc + Cost(x);
        Φ ← Σ_j p_j × max_i y_{i,j};
        if risk then
            D ← [d_i], where d_i = Σ_j p_j × e_{i,j};
            win ← j where d_j = max d_i;
            pay ← Σ_j y_{win,j} × p_j;
        else
            win ← i, where max_i(Σ_j y_{i,j} × p_j);
            pay ← Σ_j y_{win,j} × p_j;
        sav ← Φ − pay − acc;
END.
```

Figure 2. Decision process (Algorithm).

3. Identify dependencies between predicates. The dependencies are of the form predicate x_i of alternative i is true if predicate y_j of alternative j is true. For example, we may know that if the client likes alternative i, it is sure that the client also likes alternative j.
4. Delete impossible states from the set. For each relationship found, delete the states violating the relationship.

Example. Assume that there are two alternatives A_1 and A_2 for the software that implements a user interface. Assume that we are sure that our programming team can build each of them within the time and resources available (e.g., feasibility is assured for both alternatives). Therefore, the only predicates remaining are:

$cli_1, cli_2, env_1, env_2$

The possible states of nature are:

$BS_1 = cli_1 \wedge cl_2 \wedge env_1 \wedge env_2$

$BS_2 = cli_1 \wedge cli_2 \wedge env_1 \wedge \neg env_2$

$BS_3 = cli_1 \wedge cli_2 \wedge \neg env_1 \wedge env_2$

...

$BS_{16} = \neg cli_1 \wedge \neg cli_2 \wedge \neg env_1 \wedge \neg env_2$

Since the alternative A_2 is similar but more "user friendly" than A_1, we consider the following dependency:

$cli_1 \Rightarrow cli_2$

We also consider that the performance characteristics of A_2 are inferior to those of A_1; therefore, we add the dependency:

$env_2 \Rightarrow env_1$

The resulting set of states (after deleting the states violating the dependencies) is:

$S_1 = cli_1 \wedge cli_2 \wedge env_1 \wedge env_2$

$S_2 = cli_1 \wedge cli_2 \wedge env_1 \wedge \neg env_2$

$S_3 = cli_1 \wedge cli_2 \wedge \neg env_1 \wedge \neg env_2$

$S_4 = \neg cli_1 \wedge cli_2 \wedge env_1 \wedge env_2$

$S_5 = \neg cli_1 \wedge cli_2 \wedge env_1 \wedge \neg env_2$

$S_6 = \neg cli_1 \wedge cli_2 \wedge \neg env_1 \wedge \neg env_2$

$S_7 = \neg cli_1 \wedge \neg cli_2 \wedge env_1 \wedge env_2$

$S_8 = \neg cli_1 \wedge \neg cli_2 \wedge env_1 \wedge \neg env_2$

$S_9 = \neg cli_1 \wedge \neg cli_2 \wedge \neg env_1 \wedge \neg env_2$

3.3 Derivation of Payoff Matrix *Y*

Typically, each alternative will be associated with the construction of a different software solution. Depending on the state of nature that holds, the software product will become an effective solution. This degree of effectiveness determines the payoff associated with the combination alternative/state of nature. Then the questions that determine the numerical values are: How much does it cost to construct each different software solution (each alternative)? Assume we are in situation x (or in other words, assume that the state of nature is x), then how much is the client going to pay for the software solution? How much is it going to cost to adapt it to the environment? How much is going to cost to choose and build another solution if the solution we picked initially was unfeasible?

In general, the assessments we made above may not be very accurate or precise. Iterative procedures may be taken to improve the estimate. But to make the description of the decision process simple, we assume that these are accurate.

Note that if there are dominated alternatives, they should be deleted in our initial analysis. An alternative A_i is dominated by another alternative A_k iff for all states of nature S_j:

$$y_{i,j} \leq y_{k,j} \tag{11}$$

(i.e., the payoffs for A_k are greater than those for A_i for all states of nature). For example, if there were another alternative A_3 in the example of section 2, equation 1, with payoffs of (200, 200, 100), it would have been eliminated because it is dominated by A_2.

3.4 What to Prototype

The decision process function *Defineproto* defines a prototype according to the probability distribution of the states of nature. Here we give some guidelines for what to prototype to obtain the desired information.

Each state of nature s_i is represented by a boolean expression of the form:

$pred_{i,0} \wedge pred_{i,1} \wedge \ldots \wedge pred_{i,n-1}$

where each $pred_{i,j}$ is the value of a predicate j for state of nature i. Examples of predicates are client likes alternative 2, alternative 3 is feasible, alternative 4 interacts properly with environment.

Each state of nature has values true or false for n predicates.[2] Each state of nature can be represented as a binary number, where each bit corresponds to a predicate (1 for true and 0 for false). We translate the question of "what to prototype?" to "what bits to ask for in order to know which state of nature

[2] Therefore, the maximum number of states of nature is 2^n.

holds?" Making the simplification that the cost to ask for any bit is similar, the following methodology can be used:

1. Calculate the probability that the bit position i will have a 1:

   $$one_i = \sum_{k=1}^{stat} p_k \times b_{k,i}$$

   where p_k is the probability for state of nature S_k, and $b_{k,i} = 1$ if $pred_{k,i}$ = true and $b_{k,i} = 0$ if $pred_{k,i}$ = false.
2. For each bit position i calculate the absolute difference between 0.5 and one_i.

   $|0.5 - one_i|$

   This number gives a measure of the information obtained by asking for bit position i. Differences close to 0.5 correspond to positions of which values are almost known (a high probability either of 1 or 0). Differences close to 0 correspond to positions of which we are more uncertain, since the bit has almost equal chances of being 1 or 0.
3. Obtain the bit position with the minimum value of the difference calculated in the previous step. Call that position *pos*. We gain more information asking for this bit than for any other one.
4. The kind of prototype is determined by the predicate in position *pos*. Each predicate belongs to one of the categories: *cli*, *env*, *fea*. Therefore, the prototype is going to be presented to client, to environment, or will be a feasibility prototype according to the category of the predicate in position *pos*. Each predicate is associated with an alternative. The alternative associated to predicate in position *pos* is the one that is going to be prototyped. A creative (nonmechanical) decision remains: which attributes of the alternative (functionality, performance, reliability, etc.) have to be prototyped in order to decide the validity of the predicate *pos*?

A prototype defined using this methodology has two results: the predicate *pos* is true, or it is false. The designer of the prototype must estimate the probability that the prototype will answer according to the actual value of the predicate. This probability will be used to construct the conditional probability matrix C (see section 2.6).

Let $prob_{\text{true}}$ be the probability that the prototype experiment answers "predicate true" given that the predicate is true. Let $prob_{\text{false}}$ be the probability that it answers "predicate false" given that the predicate is false.

If S_j is a state in which the predicate is true, then the values of C are defined as follows:

$c_{1,j} = prob_{\text{true}}$

$c_{2,j} = 1 - prob_{\text{true}}$

If S_j is a state in which the predicate is false, then

$c_{1,j} = 1 - prob_{\text{false}}$

$c_{2,j} = prob_{\text{false}}$

Example. Assume that the vector P for the states of nature of the example of section 3.2 is:

$P = [0.3 \quad 0.2 \quad 0.1 \quad 0.1 \quad 0.08 \quad 0.07 \quad 0.05 \quad 0.05 \quad 0.05]$

The values one_i for each bit position are:

$one_1 = (p_1 + p_2 + p_3) \times 1 + (p_4 + \cdots + p_9) \times 0 = 0.6$

$one_2 = (p_1 + \cdots + p_6) \times 1 + (p_7 + p_8 + p_9) \times 0 = 0.75$

$one_3 = p_1 + p_2 + p_4 + p_5 + p_7 + p_8 = 0.78$

$one_4 = p_1 + p_4 + p_7 = 0.45$

The differences $|0.5 - one_i|$ are (0) 0.1, (1) 0.25, (2) 0.28, and (3) 0.05. The predicate with smallest difference is position (3), that is env_2. From this predicate we know that the alternative to prototype is A_2 and what is needed is to test the interactions of A_2 against the environment.

Assume that a prototype definition was designed, and the designer estimated that the prototype will determine the value of predicate env_2 with an accuracy of 95%. Now, we assign values to the elements of matrix C. For S_1, S_4, S_7 the predicate env_2 is true and we have for $j = 1, 4, 7$ as follows:

$c_{1,j} = 0.95; \; c_{2,j} = 0.05$

for the rest of the states env_2 is false, therefore for $j = 2, 3, 5, 6, 8, 9$:

$c_{1,j} = 0.05; \; c_{2,j} = 0.95$

3.5 First Estimation of the Vector *P*

In an earlier section we defined a state of nature as one of the possible combinations for the values of a set of predicates. A first approximation to derive the probabilities for the states of nature is based on the assumption that the predicates are essentially independent. We only consider the dependencies between predicates identified during the definition of the states of nature (step 3 of the methodology in section 3.2). With this simplification we define the following methodology:

1. For each predicate used in the definition of the states of nature, estimate the probability that the predicate will be true.

2. Each state of nature S_i is represented by a boolean expression of the form:
   $pred_{i,0} \wedge pred_{i,1} \wedge \ldots \wedge pred_{i,n-1}$
   Assign a probability to each $pred_{i,j}$ as follows:

- Look for a dependency of the form
  $predicate_k \Rightarrow predicate_j$
  from the dependencies identified during the definition of the states of nature. If a dependency is found and we have that
  $pred_{i,k} = true$
  then the probability associated to $pred_{i,j}$ is 1.
- if $pred_{i,j}$ = true, then use the probability estimated in step 1.
- if $pred_{i,j}$ = false, then use 1−probability estimated in step 1.

3. The probability p_i of state of nature S_i is the product of the probabilities associated with each $pred_{i,j}$.

Example. Apply the methodology to estimate P for the states of nature defined in the example of section 3.2. The probability that each predicate will be true was estimated as 0.7 for cli_1, 0.8 for cli_2, 0.9 for env_1, and 0.6 for env_2. To calculate the products we must consider the dependencies

$cli_1 \Rightarrow cli_2$; $env_2 \Rightarrow env_1$

The values for each p_i are:

$p_1 = 0.7 \times 1 \times 1 \times 0.6 = 0.42$

$p_2 = 0.7 \times 1 \times 0.9 \times (1 - 0.6) = 0.252$

$p_3 = 0.7 \times 1 \times (1 - 0.9) \times (1 - 0.6) = 0.028$

$p_4 = (1 - 0.7) \times 0.8 \times 1 \times 0.6 = 0.144$

$p_5 = (1 - 0.7) \times 0.8 \times 0.9 \times (1 - 0.6) = 0.0864$

$p_6 = (1 - 0.7) \times 0.8 \times (1 - 0.9) \times (1 - 0.6) = 0.009$

$p_7 = (1 - 0.7) \times (1 - 0.8) \times 1 \times 0.6 = 0.036$

$p_8 = (1 - 0.7) \times (1 - 0.8) \times 0.9 \times (1 - 0.6) = 0.021$

$p_9 = (1 - 0.7) \times (1 - 0.8) \times (1 - 0.9) \times (1 - 0.6) = 0.002$

Note that $\Sigma p_i = 1$. This estimation of P may be used as input for the decision process (section 3.2). It is important to take into account that this first estimation will be perfected iteratively by using prototypes.

The first step of the methodology derives the probabilities for each predicate to be true. A minimum constraint that those probabilities must follow is: If there is a dependency of the form:

$predicate_k \Rightarrow predicate_j$

then the probability that $predicate_j$ is true is greater than or equal to the probability that $predicate_k$ is true.

4. CONCLUSION AND PERSPECTIVE

To reduce the risks of software development, risk techniques for assessment and reduction from other disciplines can be applied. The adaptation and refinement of these techniques form an integral part of a quantitative theory of software management. In this article we briefly surveyed some concepts from decision theory, defined a model for a software prototype, and presented a method for applying decision theory to the problem of evaluating such prototypes.

The goal is to provide support to the software manager. The work presented here is an attempt to describe the decision process in a way that clearly separates the mechanical activities from the ones that require subjective judgement. While we have presented a mathematical model of the process, its application still depends on some subjective risk determinations by management in order to determine the appropriate probabilities that are needed by the model. We have introduced methods to help generate the input data for the decision process, namely the states of nature, their probabilities, the payoff matrix, and the prototype definitions.

In addition, we have proposed a method that helps the manager determine which form of prototype might provide the maximum information for making a decision. We have classified prototypes according to what kind of uncertainty they help to reduce—client needs, environmental needs, or project feasibility.

This model needs further refinement and evaluation of its practicality. While we believe that the overall approach of applying risk reduction strategies to software management is sound, the exact details need to be worked out. To that end, a prototype tool that applies the model developed here has been implemented and is being evaluated experimented with [15].

ACKNOWLEDGMENTS

This work was supported by Air Force Office of Scientific Research grant 90-0031 and National Science Foundation grant CCR-8819793 to the University of Maryland.

REFERENCES

1. B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.
2. R. Charette, *Software Engineering Risk Analysis and Management*, McGraw-Hill, New York, 1989.
3. M. Willis, The 10 Common Mistakes to Avoid with your Money, *Money Magazine* 19, 84–94 (1990).
4. B. W. Boehm, A Spiral Model of Development and Enhancement, *Software Eng. Notes* 11, 22–42 (1986).

5. R. Balzer, N. Goldman, and D. Wile, Operational Specifications as the Basis for Rapid Prototyping, *ACM Software Eng. Notes* 7, 3–16 (1982).
6. B. W. Boehm, T. Gray, and T. Siewaldt, Prototyping vs. specifying: A multi-project experiment, in *Proceedings of the 7th ACM/IEEE International Conference on Software Engineering*, Orlando FL 1984, pp. 433–484.
7. R. Buddle, K. Kuhlenkamp, L. Mathiassen, and H. Zallighoven, *Approaches to Prototyping*, Springer-Verlag, New York 1984.
8. S. Hekmatpour, Experience with Evolutionary Prototyping in a Large Software Project, *ACM Software Eng. Notes* 12, 38–41 (1987).
9. M. M. Tanik and R. T. Yeh, Rapid Prototyping in Software Development, *IEEE Comp.* vol. 22, No. 5, 9–10 (1989).
10. S. Cárdenas and M. Zelkowitz, Evaluation criteria for functional specifications, in *Proceedings of the 12th ACM/IEEE International Conference on Software Engineering*, Nice, France 1990, pp. 26–33.
11. F. P. Brooks, No Silver Bullet, Essence and Accidents of Software Engineering, *IEEE Comp.* vol. 20, No. 4, 10–19 (1987).
12. W. Nicholson, *Microeconomic Theory*, 3rd ed., The Dryden Press, Orlando, FL 1985.
13. J. W. Pratt, H. Raiffa, and R. Schlaifer, *Introduction to Statistical Decision Theory*, McGraw-Hill, New York, 1965.
14. M. J. Machina, Choice Under Uncertainty: Problems Solved and Unsolved, *J. Econ. Perspec.* 1, 121–154 (1987).
15. S. Cárdenas and M. Zelkowitz, A Management Tool for the Evaluation of Software Designs, IEEE Transactions on Software Engineering, 17, 9 961–971 (1991).