Chapter 10

Value Engineering of Requirements

What, Why, When, and How of Value Engineering

Up until this moment we have had very little discussion of the costs of certain features that customers may wish to include in the system requirements. Part of the reason is that it made sense to separate that very complex issue from all of the other problems surrounding requirements elicitation, analysis, representation, validation, verification, and so on. Another reason for avoiding the issue is that it is a tricky one.

In the early 1980s Barry Boehm proposed a model for software projects called the funnel curve that helped explain why software effort and cost prediction were so hard (Bohem 1981). The model had been around for some time, but Boehm first proposed its applicability to software (and later systems) projects. Steve McConnell refined the idea, calling the curve "The Cone of Uncertainty" (McConnell 1997). The model is very relevant today, and it applies both to software-only systems and complex systems that may have little or no software in them. More significantly, the model helps explain why early requirements understanding is so important, which is why we recast the model as the "Requirements Cone of Uncertainty" (Figure 10.1). It's easiest to understand the model by reading it from right to left.

Because requirements change, it is not until the project is delivered that the final list of requirements is known with certainty. Up until that time, the requirements can change. In fact, early in the life cycle of the project, at requirements definition time in particular, the requirements are highly volatile, that is, uncertain. We show

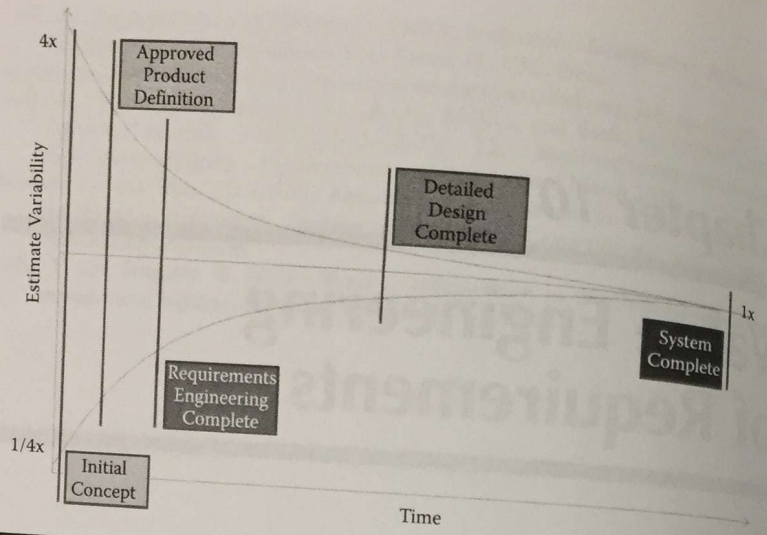

Figure 10.1 The requirements cone of uncertainty.

the estimation variability of the requirements at the project's initial conception on a scale of 4 to 1; that is, at this point in time our estimates of time to complete, and hence, cost of requirements could be as much as four times too conservative to four times too optimistic. Of course, this factor of four is entirely arbitrary.

The point of the cone model is to emphasize that it is only over time and experience that we gain precision in our ability to estimate the cost of the true requirements of the project, and so the variability of our estimates converges to their true values over time.

What Is Value Engineering?

There is a fundamental relationship between time, cost, and functionality. Project managers sometimes refer to this triad as the three legs of the project management stool. That is, you can't tip one without affecting the others. For example, you have already finished writing and agreeing to the specification and have provided an estimated cost to complete in response to some RFP (request for proposal). If the customer asks you to incorporate more features into the system, shouldn't the price and estimated time to complete increase? If not, then you were likely padding the original proposal and the customer will not like that. Conversely, if you propose to build an agreed-upon system for a price, say $1 million, and the customer balks, should you then respond by giving her a lower price, say $800,000? Of course not, because if you lower the price without reducing the feature set or increasing the time to complete, the customer will think that you gave her an inflated price

originally. The correct response, in this case, would be to agree to build the system for $800,000, but with fewer features (or taking much longer time).

When dealing with the affordability during requirements elicitation, analysis, agreement, and negotiation, both the vendor and customer have to balance certain factors. The requirements of the system determine the functional and nonfunctional features or performance of the system. A schedule has to be determined and agreed upon. The vendor and customer have to identify various risk factors in both the actual system features and in the processes to develop the system. The requirements, schedule, and risk factors determine the cost to produce, and thus, the price to the customer. The vendor is entitled to a fair profit, but at the same time, the customers' expectations need to be managed.

To properly manage customer expectations and deal with tradeoffs among functionality, time, and cost, it is therefore necessary to have some way of estimating costs for system features. Making such estimations is not easy to do accurately at the early stages of a project, such as during the requirements elicitation activities. However, it is necessary to make such cost and effort estimations. The activities related to managing expectations and estimating and managing costs are called value or affordability engineering.

When Does Value Engineering Occur?

Value engineering occurs throughout the system life cycle and is typically considered a project management activity. But for the requirements engineer value engineering has to take place to help manage customer expectations concerning the final costs of the delivered system and the feasibility or infeasibility of delivering certain features.

During the early stages of elicitation, it is probably worthwhile to entertain some value engineering activities, but not too much. For example, you may wish to temper a customer's expectations about the possibility of delivering an elaborate feature if it will break the budget. On the other hand, if you are too harsh in providing cost assessments early, you can curtail the customer's creativity and frustrate him. Therefore, be cautious when conducting value engineering when working with user-level requirements.

The best time to conduct the cost analysis is at the time when the systems-level requirements are being put together. It is at this time that better cost estimates are available, and this is a time at which tradeoff decisions can be discussed more successfully with the customer, using some of the negotiating principles previously mentioned.

A simple model then, is a reasonable place to start with an affordability engineering process for requirements is shown in Figure 10.2. First, working with the customer and other stakeholders we elicit, analyze, and create requirements using standard approaches. Next, we use some estimating techniques, such as some constructive cost modeling tool to calculate the effort needed to produce those

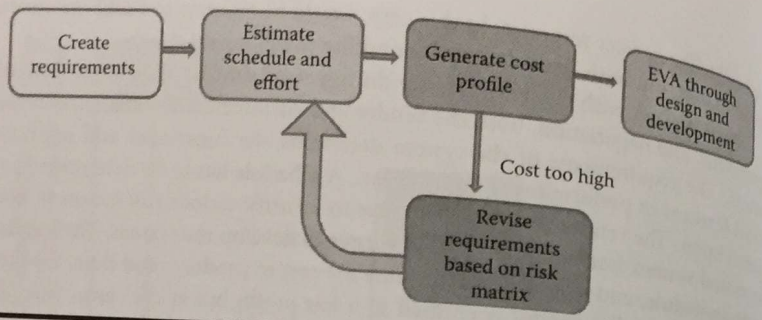

Figure 10.2 Simple affordability engineering for requirements.

requirements. The constructive cost modeling systems tool, COSYSMO, can be used to estimate the effort.

Expert opinion, analogy, Wideband Delphi, or a combination of these can be used. Next, we generate a cost profile for these requirements based on the effort estimates just calculated. If the cost of these requirements is too high, then we need to revise the requirements set based on some tradeoff analysis, for example, by using a risk matrix.

We recalculate the effort and cost for the revised requirements set until the vendor and customer are satisfied.

Finally, once the development effort commences, we can use standard project monitoring techniques, such as earned value analysis (EVA), to ensure that as requirements are built out, they meet their target cost structure. If we perceive problems with schedule or cost, the vendor and customer can work together to modify the requirements to maintain schedule and budget.

This ideal model is attractive, but there are challenges in realizing this model.

Challenges to Simple Cost versus Risk Analysis

A typical approach to balance requirements cost, risk, schedule, and so forth is to put these factors in a matrix like the one shown in Table 10.1 and employ some kind of optimization technique, even a heuristic one, to minimize cost, risk, or schedule.

But there are several problems with this somewhat simplistic representation. For example, computing true costs of requirements can be quite difficult early on, as the cone of uncertainty suggests. Furthermore, the matrix does not take into account any dependencies between requirements. These could be accounted for by adding an additional column in the matrices to list dependent requirements, but then the formulation of the problem is different. Another problem arises in the quantification of risk and identification. Different customers and stakeholders will rate risk differently. The risk matrix approach also does not account for consistency of

Table 10.1 Risk Matrix Approach to Affordability Optimization

Requirement #	Cost Estimate	Performance Metric(s)	Rank	Schedule	Risk Factor 1 = Low 5 = High
...					
3.1.2	2.5K	Response time	High	.1 SM	5
3.1.3	5K	Accuracy	Medium	.3 SM	3
3.1.4	7.5K	Accuracy	Low	.4 SM	1
...					

rating risk factors between projects. And although you can use consensus building techniques to deal with all of these problems, this is a time-consuming and, hence, expensive approach.

There are additional problems when trying to estimate the cost of requirements early in the software or system's life cycle.

It has already been noted that different customers and stakeholders will rate requirements risk differently. But they will also decompose functionality differently. For example, one customer might view a requirement for a certain style user screen to be a minor task, whereas another might view the implementation of this feature as a major undertaking.

Customers and other stakeholders have different levels of commitment to a project, and hence will invest more or less thought in their inputs. An employee who must live with any requirements decisions made will certainly behave differently than an outside consultant who does not need to cope with the decisions made during requirements analysis.

Those participating in the requirements elicitation process will have different personal involvement or "skin in the game" with respect to the eventual project outcome, and this fact will influence the quality of their participation in requirements elicitation and agreement.

Be aware, too, of the effects of differing personal agendas of the participants. For example, a customer representative who is leaving the company soon will care less about the correctness of a feature description than the prospective manager of the project.

Finally, cost-based requirements engineering can encourage individuals to set up some "plausible deniability," that is, the ability to give an excuse if the project does not go as planned. You can identify these individuals, usually, because they are unwilling to attach their names to certain documentation, or they use hedging words during requirements negotiation.

In the remainder of the chapter, we look at some simple approaches to assist in the value engineering activity for requirements engineers. These sections provide only an introduction to these very complex activities that marry project management, cost accounting, and system engineering. Really, a set of experts with these skills is needed to provide the most accurate information for decision making.

Estimating Using COCOMO and Its Derivatives*

One of the most widely used software cost and effort estimation tools is Boehm's COCOMO model, first introduced in 1981. COCOMO is an acronym for constructive cost model, which means that estimates are determined from a set of parameters characterizing the project. There are several versions of COCOMO including the original (basic), intermediate, and advanced models, each with increasing numbers of variables for tuning the estimates. The latest COCOMO models better accommodate more expressive modern languages as well as software generation tools that tend to produce more code with essentially the same effort. There are also COCOMO derivatives that are applicable for Web-based applications and software-intensive (but not pure software) systems.

COCOMO

COCOMO models are based on an estimate of lines of code, modified by a number of factors. The equations for project effort and duration are

$$\text{Effort} = A\prod_{i=1}^{9} cd_i(size)^{P_1} \tag{10.1}$$

$$\text{Duration} = B(Effort)^{P_2} \tag{10.2}$$

where A and B are a function of the type of software system to be constructed. For example, if the system is organic, that is, one that is not heavily embedded in the hardware, then the following parameters are used: $A = 3.2$, $B = 1.05$. If the system is semi-detached, that is, partially embedded, then these parameters are used: $A = 3.0$, $B = 1.12$. Finally, if the system is embedded, that is, closely tied to the underlying hardware like the visual inspection system, then the following parameters are used: $A = 2.8$, $B = 1.20$. Note that the exponent for the embedded system is the highest, leading to the longest time to complete for an equivalent number of delivered source instructions.

* Some of this section is adapted from P.A. Laplante, *Software Engineering for Image Processing Systems*. Boca Raton, FL: CRC Press, September 2003. With permission.

P_1 and P_2 are dependent on characteristics of the application domain and the cd_i are cost drivers based on a number of factors including:

- Product reliability and complexity
- Platform difficulty
- Personnel capabilities
- Personnel experience
- Facilities
- Schedule constraints
- Degree of planned reuse
- Process efficiency and maturity
- Precedentedness (i.e., novelty of the project)
- Development flexibility
- Architectural cohesion
- Team cohesion

with qualitative ratings on a Likert scale ranging from very low to very high; that is, numerical values are assigned to each of the responses.

Incidentally, effort represents the total project effort in person-months, and duration represents calendar months. These figures are necessary to convert the COCOMO estimate to an actual cost for the project.

In the advanced COCOMO models, a further adaptation adjustment factor is made for the proportion of code that is to be used in the system, namely, design modified, code modified, and integration modified. The adaptation factor, A, is given by Equation (10.3).

$$A = 0.4\ (\%\ \textit{design modified}) + 0.3\ (\%\ \textit{code modified}) + 0.3\ (\%\ \textit{integration modified}) \tag{10.3}$$

COCOMO is widely recognized and respected as a software project management tool. It is useful even if the underlying model is not really understood. COCOMO software is commercially available and can even be found on the web for free use.

An important consideration for the requirements engineer, however, is that COCOMO bases its estimation on lines of code, which are not easily estimated at the time of requirements engineering. Other techniques, such as function points, are needed to provide line of code estimates based on feature sets that are available when developing the requirements. We look at function points shortly.

WEBMO

WEBMO is a derivative of COCOMO that is intended specifically for project estimation of web-based projects, where COCOMO is not always as good.

WEBMO uses the same effort and duration equations as COCOMO, but is based on a different set of predictors, namely:

- Number of function points
- Number of xml, html, and query language links
- Number of multimedia files
- Number of scripts
- Number of web building blocks

with qualitative ratings on a Likert scale ranging from very low to very high and numerical equivalents as shown in Table 10.2 (Reifer 2002).

Similar tables are available for the cost drivers in the COCOMO model but are embedded in the simulation tools, so the requirements engineer only has to select the ratings from the Likert scale.

In any case, the net result of a WEBMO calculation is a statement of effort and duration to complete the project in person-months and calendar months, respectively.

Table 10.2 WEBMO Cost Drivers and Their Values

	Ratings				
	Very Low	*Low*	*Nominal*	*High*	*Very High*
Cost Driver					
Product reliability	0.63	0.85	1.00	1.30	1.67
Platform difficulty	0.75	0.87	1.00	1.21	1.41
Personnel capabilities	1.55	1.35	1.00	0.75	0.58
Personnel experience	1.35	1.19	1.00	0.87	0.71
Facilities	1.35	1.13	1.00	0.85	0.68
Schedule constraints	1.35	1.15	1.00	1.05	1.10
Degree of Planned					
Reuse	—	—	1.00	1.25	1.48
Teamwork	1.45	1.31	1.00	0.75	0.62
Process efficiency	1.35	1.20	1.00	0.85	0.65

Source: Adapted from D.J. Reifer, Estimating web development costs: There are differences. (2002). Online at http://www.reifer.com/download.html, last accessed March 2013.)

COSYSMO

COSYSMO (constructive system engineering model) is a COCOMO enhancement for systems engineering project cost and schedule estimation (Valerdi 2008). COSYSMO is intended to be used for cost and effort estimation of mixed hardware/software systems based on a set of size drivers, cost drivers, and team characteristics. The formulation of the COSYSMO metrics is similar to that for COCOMO, using Equations (10.1) and (10.2).

Like COCOMO, COSYSMO computes effort (and cost) as a function of system functional size and adjusts it based on a number of environmental factors related to systems engineering (Figure 10.3).

The main steps in using COSYSMO are:

1. Determine the system of interest.
2. Decompose system objectives, capabilities, or measures of effectiveness into requirements that can be tested, verified, or designed.
3. Provide a graphical or narrative representation of the system of interest and how it relates to the rest of the system.
4. Count the number of requirements in the system/marketing specification or the verification test matrix for the level of design in which systems engineering is taking place in the desired system of interest.
5. Determine the volatility, complexity, and reuse of requirements (Valerdi 2008).

In COSYSMO, the size drivers include the counts of the following items as taken right from the SRS document:

- Total system requirements
- Interfaces
- Operational scenarios
- The unique algorithms that are defined

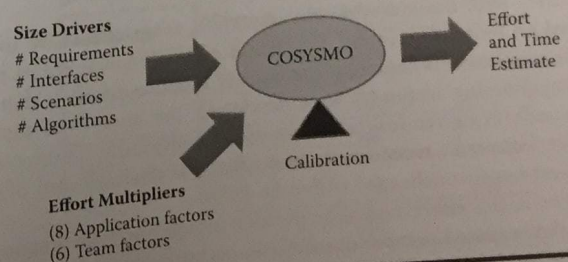

Figure 10.3 COSYSMO operational concept.

Other drivers include:

- Requirements understanding
- Architecture complexity
- Level of service requirements
- Migration complexity
- Technology maturity

which are ranked using a Likert scale in a similar manner as the COCOMO size and cost drivers.

Finally, cost drivers based on team characteristics include:

- Stakeholder team cohesion
- Personnel capability
- Personnel experience/continuity
- Process maturity
- Multisite coordination
- Formality of deliverables
- Tool support

which are also rated on a Likert scale. The decomposition of requirements into an appropriate level of detail or granularity is particularly important, but not very easy to do. An effective approach to requirements decomposition can be found in Liu, Valerdi, and Laplante (2010).

The COSYSMO model now contains a calibration dataset of more than 50 projects provided by major aerospace and defense companies such as Raytheon, Northrop Grumman, Lockheed Martin, SAIC, General Dynamics, and BAE Systems.

Estimating Using Function Points

Function points were introduced in the late 1970s as an alternative to product metrics based on source line count. This aspect makes function points especially useful to the requirements engineer. The basis of function points is that as more powerful programming languages were developed, the number of source lines necessary to perform a given function decreased. Paradoxically, however, the cost/LOC measure indicated a reduction in productivity, as the fixed costs of software production were largely unchanged (Albrecht 1979).

The solution to this effort estimation paradox is to measure the functionality of software via the projected number of interfaces between modules and subsystems in programs or systems. A big advantage of the function point metric is that it can be calculated during the requirements engineering activities.

Function Point Cost Drivers

The following five software characteristics for each module, subsystem, or system represent the function points or cost drivers:

- Number of inputs to the application (I)
- Number of outputs (O)
- Number of user inquiries (Q)
- Number of files used (F)
- Number of external interfaces (X)

In addition, the FP calculation takes into account weighting factors for each aspect that reflect their relative difficulty in implementation, and the function point metric consists of a linear combination of these factors, as shown in Equation (10.4).

$$FP = w_1 I + w_2 O + w_3 Q + w_4 F + w_5 X \tag{10.4}$$

where the w_i coefficients vary depending on the type of application system. Then complexity factor adjustments are applied for different types of application domains. The full set of coefficients and corresponding questions can be found by consulting an appropriate text on software metrics. The International Function Point Users Group maintains a web database of weighting factors and function point values for a variety of application domains.

For the purposes of cost and schedule estimation and project management, function points can be mapped to the relative lines of source code in certain programming languages. A few examples are given in Table 10.3.

Now, these lines of code counts can be plugged into the COCOMO estimation equations to obtain appropriate estimates of effort for various features.

Table 10.3 Programming Language and Lines of Code per Function Point

Language	Lines of Code per Function Point
C	128
C++	64
Java	64
SQL	12
Visual Basic	32

Source: Adapted from Jones, *IEEE Computer*, May, 103–104 (1996).

Here is an example of how this might work. A customer asks for a cost estimate for a desired feature. Based on the details of that feature, and the various weighting factors needed to calculate FP, the FP metric is computed. That number is converted to lines of code count using the conversion shown in Table 10.2 (or another, appropriate conversion). These lines of code count number, along with the various other aspects of the project, are plugged into a COCOMO estimator, which yields an estimate of time and effort (meaning person-hours) to complete the project. Of course, you wouldn't take this estimate for granted. It would be appropriate to check the estimate using other techniques that are complementary to the FP/COCOMO estimation approach. In any case, let's assume that you believe the estimate that COCOMO gives you. This estimate of number of person-months can be converted into an appropriate cost estimate for the customer, a task that would probably be done through the sales department.

Feature Points

Feature points are an extension of function points developed by Software Productivity Research, Inc. in 1986. Feature points address the fact that the classical function point metric was developed for management information systems and therefore are not particularly applicable to many other systems, such as real-time, embedded, communications, and process control software. The motivation is that these systems exhibit high levels of algorithmic complexity, but sparse inputs and outputs.

The feature point metric is computed in a similar manner to the function point except that a new factor for the number of algorithms, A, is added, yielding Equation (10.5).

$$FP' = w_1 I + w_2 O + w_3 Q + w_4 F + w_5 X + w_6 A \tag{10.5}$$

Use Case Points

Use case points (UCP) allow the estimation of an application's size and effort from its use cases. This is a particularly useful estimation technique during the requirements engineering activity when use cases are the basis for requirements elicitation.

The use case point equation is based on the product of four variables that are derived from the number of actors and scenarios in the use cases and various technical and environmental factors. The four variables are:

1. Technical Complexity Factor (TCF)
2. Environment Complexity Factor (ECF)
3. Unadjusted Use Case Points (UUCP)
4. Productivity Factor (PF)

leading to the basic use case point equation:

$$UCP = TCF \cdot ECF \cdot UUCP \cdot PF \tag{10.6}$$

This metric is then used to provide estimates of project duration and staffing from data collected from other projects. As with function points, additional technical adjustment factors can be applied based on project, environment, and team characteristics (Clemmons 2006).

Artificial neural networks and stochastic gradient boosting techniques (Nassif et al. 2012) have been used to enhance the predictive accuracy of use case points. Use case points are clearly beneficial for effort estimation in those projects where the requirements are created predominantly via use cases.

Requirements Feature Cost Justification

Consider the following situation. A customer has the option of incorporating feature A into the system for $250,000 or forgoing the feature altogether. Currently $1,000,000 is budgeted for activity associated with feature A. It has been projected that incorporating the feature into the software would provide $500,000 in immediate cost savings by automating several aspects of the activity.

Should the manager decide to forgo including feature A into the system, new workers would need to be hired and trained before they could take up the activities associated with feature A.* At the end of two years, it is expected that the new workers would be responsible for $750,000 cost savings. The value justification question is, "Should the new feature be included in the system?" We can answer this question after first discussing several mechanisms for calculating the value of certain types of activities, including the realization of desired features in the system.

Return on Investment

Return on investment (ROI) is a rather overloaded term that means different things to different people. In some cases it means the value of the activity at the time it is undertaken. To others it is the value of the activity at a later date. In other cases it is just a catchword for the difference between the cost of the system and the savings anticipated from the utility of that system. To still others there is a more complex meaning.

One traditional measure of ROI for any activity or system is given as

$$ROI = \text{Average Net Benefits/Initial Costs} \tag{10.7}$$

* Such a cost is called a "sunken cost" because the money is gone whether one decides to proceed with the activity or not.

The challenge with this model for ROI is the accurate representation of average net benefit and initial costs. But this is an issue of cost accounting that presents itself in all calculations of costs versus benefits.

Other commonly used models for valuation of some activity or investment include net present value (NPV), internal rate of return (IRR), profitability index (PI), and payback. We look at each of these shortly.

Other methods include Six Sigma and proprietary Balanced Scorecard models. These kinds of approaches seek to recognize that financial measures are not necessarily the most important component of performance. Further considerations for valuing software solutions might include customer satisfaction, employee satisfaction, and so on, which are not usually modeled with traditional financial valuation instruments.

There are other, more complex accounting-oriented methods for valuing software. Discussion of these techniques is beyond the scope of this text. The references at the end of the chapter can be consulted for additional information (e.g., Raffo, Settle, and Harrison 1999; Morgan 2005).

Net Present Value

Net present value is a commonly used approach to determine the cost of software projects or activities. The NPV of any activity or thing is based on the notion that a dollar today is worth more than that same dollar tomorrow. You can see the effect as you look farther into the future, or back to the past (some of you may remember when gasoline was less than $1 per gallon). The effect is due to the fact that, except in extraordinary circumstances, the cost of things always escalates over time.

Here is how to compute NPV. Suppose that FV is some future anticipated payoff either in cash or in anticipated savings. Suppose r is the discount rate,* and Y is the number of years that the cash or savings is expected to be realized. Then the net present value of that payoff is:

$$\text{NPV} = FV/(1 + r)^Y \tag{10.8}$$

NPV is an indirect measure because you are required to specify the market opportunity cost (discount rate) of the capital involved.

To see how you can use this notion in value engineering of requirements, suppose that you expect a certain feature, B, to be included in the system at a cost of $60,000. You believe that benefits of this feature are expected to yield $100,000 two years in the future. If the discount rate is 3%, should the feature be included in the new system?

* The interest rate charged by the US Federal Reserve. The cost of borrowing any capital will be higher than this base rate.

To answer this question, we calculate the NPV of the feature using Equation (10.8), taking into account the cost of the feature:

$$\text{NPV} - \text{cost} = 100{,}000/1.03^2 - 60{,}000 = 34{,}259$$

Because NPV less its cost is positive, yes, the feature should be included in the requirements.

Equation (10.8) is useful when the benefit of the feature is realizable after some discrete period of time. But what happens if there is some incremental benefit of adding a feature; for example, after the first year, the benefit of the feature doubles (in future dollars). In this case we need to use a variation of Equation (10.8) that incorporates the notion of continuing cash flows.

For a sequence of cash flows, CF_n, where $n = 0, \ldots, k$ represents the number of years from initial investment, the net present value of that sequence is

$$\text{NPV} = \sum_{n=0}^{k} \frac{CF_n}{(1+r)^n} \tag{10.9}$$

The CF_n could represent, for example, a sequence of related expenditures over a period of time, such as features that are added incrementally to the system or evolutionary versions of a system.

Internal Rate of Return

The internal rate of return (IRR) is defined as the discount rate in the NPV equation that causes the calculated NPV, minus its cost, to be zero. This can be done by taking Equation (10.8) and rearranging to obtain

$$0 = FV/(1 + r)^Y - c$$

where c is the cost of the feature and FV its future value. Solving for r yields

$$r = [FV/c]^{1/Y} - 1 \tag{10.10}$$

For example, to decide if we should incorporate a feature, we compare the computed IRR of including the feature with the financial return of some alternative, for example, to undertake a different corporate initiative or add a different feature. If the IRR is very low for adding the new feature, then we might simply want to take this money and find an equivalent investment with lower risk. But if the IRR is sufficiently high for the new feature, then the decision might be worth whatever risk is involved in its implementation.

To illustrate, suppose that a certain feature is expected to cost $50,000. The returns of this feature are expected to be $100,000 of increased output two years in the future. We would like to know the IRR on adding this feature.

Here, NPV – *cost* = $100{,}000/(1 + r)^2 - 50{,}000$. We now wish to find the r that makes the NPV = 0, that is, the "break even" value. Using Equation (10.10) yields

$$r = [100{,}000/50{,}000]^{1/2} - 1$$

This means $r = 0.414 = 41.4\%$. This rate of return is very high for this feature, and we would likely choose to incorporate it into the system.

Profitability Index

The profitability index (PI) is the NPV divided by the cost of the investment, I:

$$PI = NPV/I \quad (10.11)$$

PI is a "bang-for-the-buck" measure, and it is appealing to managers who must decide among many competing investments with positive NPV financial constraints. The idea is to take the investment options with the highest PI first until the investment budget runs out. This approach is not bad but can suboptimize the investment portfolio.

One of the drawbacks of the profitability index as a metric for making resource allocation decisions is that it can lead to a suboptimization of the result. To see this effect, consider the following situation. A customer is faced with some hard budget-cutting decisions, and she needs to remove some features from the proposed system. To reach this decision she prepares the analysis based on the profitability index shown in Table 10.4.

Suppose the capital budget is $500K. The PI ranking technique would cause the customer to pick A and B first, leaving inadequate resources for C. Therefore, D will be chosen, leaving the overall NPV at $610K. However, using an integer

Table 10.4 A Collection of Prospective Software Features, Their Project Cost, NPV, and Profitability Index

Feature	Projected Cost (in $10s of thousands)	NPV (in $10s of thousands)	PI
A	200	260	1.3
B	100	130	1.3
C	300	360	1.20
D	200	220	1.1

programming approach will lead to a better decision (based on maximum NPV) in that features A and C would be selected, yielding an expected NPV of $620K.

The profitability index method is helpful in conjunction with NPV to help optimize the allocation of investment dollars across a series of feature options.

Payback Period

To the project manager, the payback period is the time it takes to get the initial investment back out of the project. Projects with short paybacks are preferred, although the term "short" is completely arbitrary. The intuitive appeal is reasonably clear: the payback period is easy to calculate, communicate, and understand.

Payback can be used as a metric to decide whether to incorporate a requirement in the system. For example, suppose implementing feature D is expected to cost $100,000 and result in a cost savings of $50,000 per year. Then the payback period for the feature would be two years.

Because of its simplicity, payback is the least likely ROI calculation to confuse managers. However, if payback period is the only criterion used, then there is no recognition of any cash flows, small or large, to arrive after the cutoff period. Furthermore, there is no recognition of the opportunity cost of tying up funds. Because discussions of payback tend to coincide with discussions of risk, a short payback period usually means a lower risk. However, all criteria used in the determination of payback are arbitrary. And from an accounting and practical standpoint, the discounted payback is the metric that is preferred.

Discounted Payback

The discounted payback is the payback period determined on discounted cash flows rather than undiscounted cash flows. This method takes into account the time (and risk) value of the sunken cost. Effectively, it answers the questions, "How long does it take to recover the investment?" and, "What is the minimum required return?"

If the discounted payback period is finite in length, it means that the investment plus its capital costs are recovered eventually, which means that the NPV is at least as great as zero. Consequently, a criterion that says to go ahead with the project if it has *any* finite discounted payback period is consistent with the NPV rule.

To illustrate, in the payback example just given, there is a cost of $100,000 and an annual maintenance savings of $50,000. Assuming a discount rate of 3%, the discounted payback period would be longer than two years because the savings in year two would have an NPV of less than $50,000 (figure out the exact payback period for fun). But because we know there is a finite discounted payback period, we know that we should go ahead and include feature D.

Putting It All Together

Suppose the smart home were being built for some special customer. This customer presumably conducted his own ROI analysis for this system using one of the methods previously introduced and now understands how much he would like to pay for the system (or even parts of the system). For example, this was a one-of-a-kind home (think of Bill Gates's smart home) therefore the investment in this functionality would need to add at least as much as the cost of the features to the resale price.

Let's say that the homeowner felt that this feature would add $250K to the asking price of his resold home, and he has decided that he is happy with an ROI of 1.25. Therefore, he would be willing to pay $200K for these features.

Suppose we analyzed these requirements and used COSYSMO to derive a cost estimate of $210K. The customer may indicate that he is only willing to pay $200K. In order to lower the price, we can suggest eliminating certain requirements in order to meet his budgetary constraints.

For example, suppose we omit requirements 8.1.10 and 8.1.13, then get a new estimate of $200K, and COSYSMO predicts that it will to take 10 calendar months to build and that the customer can resell the system immediately upon completion. Now we can compute a profitability index as well as an internal rate of return.

Let's assume a discount rate of 3%. The NPV of this activity, using $250K over 10 months at 3% (using simple interest) is

$$NPV = \$250{,}000*(1.03)10/12 - \$200{,}000 = \$43{,}917.14.$$

Continuing, the profitability index on this set of requirements is

$$PI = \$256{,}234/\$200{,}000 = 1.2811$$

The internal rate of return on this investment is

$$IRR = [\$250{,}000/\$200{,}000]12/10 - 1 = 30.7\%$$

Thus we resolved a cost issue for our customer by revising the effort and generated a good investment for him.

Exercises

10.1 Why is it so important to determine the cost of features early, but not too early in the requirements engineering process?

10.2 What factors determine which metric or metrics a customer can use to help make meaningful cost-benefit decisions of proposed features for a system to be built?

10.3 How does the role of ranking requirements help in feature selection cost-benefit decision making?

10.4 What changes (if any) would you need to make to the COCOMO, COSYSMO, or feature/function point equation calculations to incorporate ranking of requirements?

10.5 Complete the derivation of Equation (10.8) from Equation (10.6) by setting the NPV equation (less the cost of the investment) to zero and solving for *r*.

*10.6 Investigate the use of other decision-making techniques, such as integer programming, in helping to decide on the appropriate feature set for a proposed system.

References

Albrecht, J. (1979). Measuring application development productivity. In *Proceedings of the IBM Applications Development Symposium*, Monterey, CA, Oct. pp. 14–17.

Boehm, B.W. (1981). *Software Engineering Economics*. Upper Saddle River, NJ: Prentice Hall.

Clemmons, R.K. (2006). Project estimation with use case points, *Journal of Defense Software Engineering*, (February) 18–22.

Jones, C. (1996). Activity-based software costing, *IEEE Computer*, May, 29(5): pp. 103–104.

Laplante, P.A. (2006). *What Every Engineer Needs to Know About Software Engineering*. Boca Raton, FL: CRC/Taylor & Francis.

Laplante, P.A. (2003). *Software Engineering for Image Processing Systems*. Boca Raton, FL: CRC Press.

Liu, K., Valerdi, R., and Laplante, P. A. (2010). Better requirements decomposition guidelines can improve cost estimation of systems engineering and human systems integration. In *8th Annual Conference on Systems Engineering Research*. Hoboken, NJ.

McConnell, S. (1997). *Software Project Survival Guide*, Redmond, WA: Microsoft Press.

Morgan, J.N. (2005). A roadmap of financial measures for IT project ROI. *IT Professional*, Jan./Feb., pp. 52–57.

Nassif, A.B., Capretz, L.F., and Ho, D. (2012). Estimating software effort using an ann model based on use case points. In *11th International Conference on Machine Learning and Applications* (ICMLA), December , Vol. 2. Washington, DC: IEEE. pp. 42–47.

Nassif, A.B., Capretz, L.F., Ho, D., and Azzeh, M. (2012). A Treeboost model for software effort estimation based on use case points. In *11th International Conference on Machine Learning and Applications* (ICMLA), December, Vol. 2. Washington, DC: IEEE. pp. 314–319.

Raffo, D., Settle, J., and Harrison, W. (1999). *Investigating Financial Measures for Planning of Software IV&V*, Portland State University Research Report #TR-99-05.

Reifer, D.J. (2002). Estimating web development costs: There are differences. Online at http://www.reifer.com/download.html, last accessed March 2013.

Valerdi, R. (2008). *The Constructive Systems Engineering Cost Model (COSYSMO): Quantifying the Costs of Systems Engineering Effort in Complex Systems*. Saarbrücken, Germany: VDM Verlag.

* This exercise is suitable for a small research assignment.