Business Cases That Make Sense

The true speculator is one who observes the future and acts before it occurs. Like a surgeon, he must be able to search through a mass of complex and contradictory details to the significant facts. Then, like the surgeon, he must be able to operate coldly, clearly, and skillfully on the basis of the facts before him.

—Bernard Baruch [1957]

This chapter introduces you to the case studies that follow in Chapters 5 through 8. It sets the stage and provides you with the necessary background information you need to understand what the case studies are all about.

The business planning process outlined in Chapter 2 (see Figure 2.4) will be the starting point for most of you. You will kick off your effort by writing a white paper justifying your idea or improvement using some tangible benefits. In it, you will set your goals and discuss how you will achieve them. The paper will allow you to set aggressive but realizable expectations. Because most of you will try to justify your proposals using productivity improvement, cost avoidance, or time-to-market decreases, I open this chapter with some additional guidance on what to do when using these measures. In the remainder of this chapter, I provide you with background material on the four case studies that follow (see Chapters 5–8) so that you will understand them more completely as we go through the remainder of the business planning process.
THE PARABLE OF THE CHINESE EMPEROR

Let's start by looking at software productivity and ways to improve it. When I think of productivity, I think of the parable of the Chinese emperor. The emperor wanted to optimize productivity as his subjects dug a tunnel through a mountain. He asked his chief advisor how to achieve this goal. This wise, old man said: "Form two teams, and have them dig toward each other from either side of the mountain." He then amplified: "Because they will meet in the middle, productivity will be increased." The emperor thought for a minute. Then, he stated in an alarmed manner: "What happens if the teams don't meet?" The sage quickly replied: "Productivity will increase as well because two tunnels will result." The emperor peremptively replied: "Building tunnels is a win-win situation."

This parable illustrates the dual nature of productivity. Because productivity is defined as the ratio of outputs to inputs used to generate them, implementing strategies aimed at either side of the equation can optimize it. To improve the input side of the formula, organizations try to get the most they can from their employees by arming them with good processes, methods, tools, and a modern workspace. To optimize the output side, they attempt to make large jobs smaller using application generators, reuse, and component-based software engineering strategies. Both techniques work. However, better results occur when both techniques are put into action at the same time.

In any case, productivity improvement requires a strategy. My advice is to start by tackling the simple and obvious targets of opportunity first. Productivity improvement in virtually all organizations is there for the taking. All you have to do is search for the untapped and underutilized resources to make a positive impact. The sum of many small individual improvements can add up to significant increases. That's why I recommend pursuing many small changes in parallel instead of a single big change all at once. While the net result is the same, small improvements are much more manageable. In addition, a relatively small increase in overall productivity can frequently have a large impact on profit as long as a proper perspective and a reasonable pace are maintained.

Most people use productivity to justify their initiatives. Guidelines recommending what to do and what not to do when using productivity as the basis of your justification for improvements are provided in Table 4.1. These guidelines assume that there is no mismatch between how your firm defines organizational productivity and software productivity. If there is, you need to fix this problem by somehow relating your definition to that used by your organization. Your goal
### Table 4.1: Productivity Improvement Guidelines

<table>
<thead>
<tr>
<th>What to Do</th>
<th>What Not to Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>Be prepared to deliver what you promise; you will be held accountable for results.</td>
<td>Don’t forget to make sure you align your proposals with your firm’s business objectives.</td>
</tr>
<tr>
<td>Use definitions for productivity that are accepted by your industry and your firm.</td>
<td>Don’t make promises you don’t think you can keep.</td>
</tr>
<tr>
<td>If you don’t know what it is, benchmark your current software productivity.</td>
<td>Don’t leave anything to chance. Do your homework when it comes to numbers and know what they mean and how they are derived mathematically.</td>
</tr>
<tr>
<td>Benchmark the competition’s productivity. Ask customers to rate best of class if the data is not readily available to you.</td>
<td>Don’t use terms and definitions that are foreign to your company and industry.</td>
</tr>
<tr>
<td>Plot a trend line to compare the current rate of software productivity improvement made by your firm to industry norms.</td>
<td>Don’t assume that everyone knows what your current software productivity is and how it is measured. I can assure you that most don’t.</td>
</tr>
<tr>
<td>Use ranges instead of discrete numbers to bound your productivity numbers.</td>
<td>Don’t assume you can find out what your competition’s productivity is. The best you can probably do is get customer perceptions about who is best in class.</td>
</tr>
<tr>
<td>Define what your numbers include and don’t include.</td>
<td>Don’t forget to double- and triple-check your numbers before you present them. A math error can lead to an immediate and irrecoverable credibility gap.</td>
</tr>
<tr>
<td>Prepare backup materials to defend your justification; if management is interested, they will ask you to supply details in a follow-up briefing.</td>
<td></td>
</tr>
</tbody>
</table>

is to show management how proposed improvements in software productivity contribute to the bottom line using definitions they are familiar with and understand.

### PROCESS IMPROVEMENT USING PRODUCTIVITY INCREASES AS JUSTIFICATION

In Chapter 5, I present a case study that justifies pursuing process improvement by productivity improvement. Process improvement in this case is defined in terms of the Software Engineering Institute’s Software Capability Maturity Model (SW-CMM) [Paulk, 1995]. The SW-CMM provides a conceptual framework for...
improving how you manage and develop your software products. It is organized around five levels of maturity that you can use to prioritize the opportunities that exist for process improvement. Clusters of related activities that, when performed, achieve the goals established for improving process capabilities at each

**Level 5—Optimizing**
- Defect prevention
- Technology change management
- Process change management

**Process Characteristics**
- Processes are continually and systematically improved.
- Common sources of problems are understood and eliminated.

**Level 4—Managed**
- Quantitative software management
- Software quality management

- Processes are quantitatively understood.
- Focus is on minimizing process variability via statistical process control techniques.

**Level 3—Defined**
- Organization process focus
- Organization process definition
- Training program
- Integrated software management
- Software product engineering
- Intergroup coordination
- Peer review

- Processes are used in common across the organization.
- Focus is on teamwork and tailoring the process to work.
- Metrics are collected and used to improve performance.
- Groups work together in integrated product teams.
- Problems are anticipated and their impacts are minimized.

**Level 2—Repeatable**
- Requirements management
- Software project planning
- Software project tracking and control
- Software subcontract management
- Software quality assurance
- Software configuration management

- Documented and stable set of processes is used at the project level.
- Focus is on establishing project management infrastructure.

**Level 1—Ad hoc**

**Increasing Levels of Maturity**

![Figure 4.1: Characteristics and Key Process Areas of Maturity Levels](source.png)
level of the framework are called key process areas (KPA). The five levels are illustrated along with their KPAs in Figure 4.1. Although the software framework is being incorporated into the new CMM integration (CMMI), this example represents an interesting case study because it shows how just a little productivity data can be used to justify the large investments in infrastructure needed to pull it off. I plan to replace this case with one that emphasizes the CMMI when a relevant example becomes available.

During the past decade, many organizations have provided reports documenting their use of the SW-CMM to structure the processes that software organizations use to generate their products. A process group conference sponsored by the SEI is held every year, and monthly Software Process Improvement Network (SPIN) meetings are held throughout the world. In addition, the number of organizations embarking on process improvement efforts seems to increase every year. Unfortunately, most of the experience reports I've read from these organizations, while informative and valuable, have been qualitative in nature. They discuss the experience and benefits that accrue due to the use of the CMM in soft rather than hard terms.

In the early days of process improvement, most of the organizations that adopted the maturity framework were aerospace/military. However, that tendency has changed in recent years; by far, most new organizations are commercial. The current demographics taken from 1,166 organizations who have conducted 1,512 assessments on 6,168 projects are illustrated in Figures 4.2 and 4.3 [SEI, 2000].

![Figure 4.2: Organizational Ratings by Maturity Level](image)

Source: Figures 4.2 and 4.3 adapted from Software Engineering Institute (SEI), Process Maturity Profile of the Software Community 1999 Year-End Update, March 2000. Special permission to reproduce and use portions of the Carnegie Mellon University publication, © 2000, is granted by the Software Engineering Institute. CMM® and Capability Maturity Model are registered in the U.S. Patent and Trademark Office.
The following compilation of hard data on benefits summarizes the facts that I have been able to find in the open literature relative to process improvement:

- On average, it takes between 18 and 30 months to move from one maturity level to another. The median times to shift toward higher maturity are as follows [SEI, 2000]:
  - From Level 1 to Level 2: 25 months
  - From Level 2 to Level 3: 23 months
  - From Level 3 to Level 4: 36 months

- The average investment to move up one such level in maturity rating is hefty, in the millions [Haley, 1995]. These costs often fund operation of a process group whose job is to write and deploy the process that fits in the CMM framework [Harp, 2001].

- The gains attributable to early error detection and correction can be substantial, assuming that it is 20 times cheaper to fix an error found during the requirements and design phase than during test and integration [Harp, 2001].

- The average increase in productivity attributable to process improvement is about 10 percent [Harp, 2001].

The most detailed study of benefits associated with the CMM that I have seen is in the Ph.D. thesis of Dr. Brad Clark for the University of Southern California. Using a 161 project database, he concluded that an increase in one process maturity level can reduce development effort by between 4 to 11 percent [Clark, 2000].
COST AVOIDANCE VERSUS COST REDUCTION

Many people equate productivity improvement with either cost reduction or cost avoidance. This is not always the case: in many situations, increasing productivity results in increased cost. Take the case of teams producing software to the wrong requirements. While they may be very productive, rework and false starts due to requirements evolution and volatility could force them to throw away much of what they produced. Taking one step backward to get their requirements right might be a better alternative.

As another example, consider the productivity of people who fail to devote effort early in the development to defect removal. Their productivity would seem high until the poor quality caught up with them either during testing or after the product was released. Because improvements in one dimension don’t always lead to improvements in the other, I suggest separating cost and productivity when generating an improvement justification. I also suggest not including profit in your numbers when justifying an improvement using a cost avoidance. There are situations where you can take a loss on a job and still make a profit. You might, for example, simply price your product at less than its cost to capture market share (e.g., as a loss leader). Your market entry strategy would be to keep the purchase price low and make your money on maintenance or license renewals. Many software firms use this strategy to reap large profits. They sell their products cheaply and make their money on annual releases or updates.

There may be other sound justifications for taking a loss on a job. For example, the economics of the situation may show that doing the job reduces overhead and thereby results in increased profit. For example, let’s say that your software department is working at 80 percent capacity (i.e., charging 80 percent of their work to projects). A job comes in that you can take for a loss that will allow you to charge the remaining 20 percent of the workforce to a project. This job could then result in a profit because it would take people off overhead and put them to work for a client. It could also help make the firm more competitive because reductions in overhead could reduce the price a client would pay for other goods and services.

This discussion suggests that things aren’t always what they seem when looking at profits. But costs are costs. They have to be accounted for somehow in your ledgers. If they aren’t, the auditors will be upset.

The most important thing to know when developing improvement justifications is the money system. As I stated in Chapters 2 and 3, knowing it
enables you to know when surplus money will become available. In addition to being creative with how you allocate these budgets, you need to understand how your proposal affects them. Let me illustrate this point using the training example I introduced in Chapter 2. Unfortunately, the budget surplus that was found is no longer available. It was used to bail a project out of trouble. You examine other budgets and find a line item called university relationships funded at $250,000 per year. Its aim is to improve recruiting of new college graduates via grants to foster improved relationships with local colleges and universities. Further investigation indicates that half of this year’s allocation has not been spent. Proposing to take $25,000 to fund your shortfall seems logical. However, when queried, your champion warns you against pursuing such use. He says that the president of the company has earmarked the money for his alma mater. You would probably ruin your career if you tried to grab it. But you have a brainstorm. Why not fund the university to provide the training you need with this money? The school would get the money, and you would get your training. Of course, you would have to check to make sure that the university could deliver the needed training. But, it’s an opportunity that you can seize.

When justifying improvements, cost avoidance is preferable to cost reduction. That’s because reductions occur now, while avoidance takes place in the future. Cost avoidance works best when you are in a growth mode because you can offset costs using increased demand or sales. For example, you could justify your training proposal based on decreased hiring instead of reduced turnover. Both are reasonable justifications. Both approaches look to the future. Both request additional funds without jeopardizing current budgets and staff retention.

I believe you need to focus your attention on controlling the cost drivers that are under your control. For example, you can control the experience of the personnel you assign to a project. However, you can’t control the project’s requirements because others define their content. While you can influence and manage the requirements, content is outside your control. This principle of controlling the controllables is important because it focuses your attention on things you can do to make things happen without someone else approving them. Management has a feel for the numbers. Their perceptions are often based on hearsay and folklore, not hard data. However, you need to respect their convictions to stay out of trouble.

I remember trying to convince executives at a semiconductor firm to capitalize their software development operation by increasing the per-seat investment in software engineering tools and equipment. My pitch was going fine
Table 4.2: Cost Avoidance Guidelines

<table>
<thead>
<tr>
<th>What to Do</th>
<th>What Not to Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Whenever possible, use cost avoidance, not cost reductions, to justify your improvement.</td>
<td>• Don’t assume that your numbers will not be scrutinized.</td>
</tr>
<tr>
<td>• Separate cost considerations and productivity considerations.</td>
<td>• Don’t suppose that everyone knows what your current costs are and how they are allocated to different cost centers.</td>
</tr>
<tr>
<td>• Know your money system, including when budget surpluses become available and how to tap them.</td>
<td>• Don’t confuse management by putting cost and profit considerations in the same proposal.</td>
</tr>
<tr>
<td>• Know how your current costs are normally allocated to different cost centers. Understand the sensitivity of your cost justifications to different cost drivers.</td>
<td>• Don’t pursue improvements that are outside your immediate span of control.</td>
</tr>
<tr>
<td>• Know what costs you can control and who controls the others.</td>
<td>• Don’t mix cost accounts when justifying your improvement proposal. Keep labor separate from other costs; never mix project and capital budgets.</td>
</tr>
<tr>
<td>• Offset costs against growth trends to get your proposals funded.</td>
<td>• Don’t assume that management will take your numbers at face value (i.e., respond to what they perceive the numbers should be even if they are wrong).</td>
</tr>
<tr>
<td>• Package your numbers using the guidance provided in Chapters 1, 2, and 3.</td>
<td></td>
</tr>
</tbody>
</table>

until I showed the dollar cost per seat for hardware versus software engineer. Management wouldn’t believe the disparity. Their perception was that hardware and software capitalization costs were equivalent. Of course, they weren’t. But I had to pull out my backup charts that showed the detailed cost breakouts per engineer before I could get past that point in the presentation.

Table 4.2 provides guidelines of what to do and what not to do when using cost avoidance as the basis of your justification for your idea or improvements. They assume that your numbers are sound and that those who review them will not find any mathematical errors.

SOFTWARE CAPITALIZATION APPROACHES

In Chapter 6, I present a case study that justifies capitalizing software using cost avoidance as the basis. Capitalization of software involves spending money to save
money. To get the most from your people, like the Chinese emperor, you would provide them with a well-equipped and modern software engineering environment:

- **Equipment**  Servers, workstations, printers, scanners, test gear, and access to copiers
- **Facilities**  Personal workspace, meeting rooms, training rooms, and labs
- **Networks**  Local and wide area networks connecting equipment and facilities and tying them to the Internet to allow teams of geographically dispersed engineers to collaborate and build software products
- **Software tools**  An integrated collection of software programs that automate selected “best practices” and that make work products available from some repository under some form of access control to all the team members working on the project
- **Components**  Libraries of building blocks and COTS packages used to build applications
- **Communications gear**  Routers, switches, telephones (wireless, portables, and so on), fax machines, and specialized software that enable engineers to send/receive voice, data, video, and multimedia information to each other across networks

To develop such an environment, you must take into account the differences between yesterday’s and today’s corporations as highlighted in Table 4.3. I remember how fascinated we in America were when the Japanese built the first software factories as strategic resources for developing software [Cusumano, 1987]. They recognized well before we did that software should be capitalized. Capitalization in this sense means that senior management recognizes that a large part of their capital budget must be used to acquire a modern environment for building software. Without it, they cannot generate products for today’s marketplace.

Software development in the twenty-first-century view of the corporation summarized in Table 4.3 will have a profound effect on software environment design. This distributed view of development assumes that the environment will include equipment and facilities that link geographically dispersed workforces, provide Web-based tools, support collaboration, address mobile code development, support distributed testing via agents, handle version control and documentation automatically, and provide support for prototyping and experimentation.

It is interesting to look at what it costs to acquire this kind of environment. Information displayed in Table 4.4 sheds some light on what the underlying...
Table 4.3: Contrasting Views of the Corporation

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Twentieth Century</th>
<th>Twenty-first Century</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goals</td>
<td>Grow the business nationally</td>
<td>Grow the business internationally</td>
</tr>
<tr>
<td>Organization</td>
<td>Hierarchical</td>
<td>Collaborative</td>
</tr>
<tr>
<td>Markets</td>
<td>National</td>
<td>The World</td>
</tr>
<tr>
<td>Structure</td>
<td>Self-reliance</td>
<td>Global partnerships</td>
</tr>
<tr>
<td>Emphasis</td>
<td>Physical plant</td>
<td>Corporate infrastructure</td>
</tr>
<tr>
<td>Source of strength</td>
<td>People</td>
<td>Process and team</td>
</tr>
<tr>
<td>Strategy</td>
<td>Maximize efficiency</td>
<td>Be agile and quick to market</td>
</tr>
<tr>
<td>Tactics</td>
<td>High productivity and low cost</td>
<td>E-everything</td>
</tr>
<tr>
<td>Competition</td>
<td>Corporate teams</td>
<td>National groupings</td>
</tr>
<tr>
<td>Products</td>
<td>Product families</td>
<td>Product lines</td>
</tr>
<tr>
<td>Technology</td>
<td>Manufacturing-oriented</td>
<td>Consumer-oriented</td>
</tr>
<tr>
<td>Workforce</td>
<td>Domestic</td>
<td>International</td>
</tr>
<tr>
<td>Expectations</td>
<td>Security</td>
<td>Mobility</td>
</tr>
<tr>
<td>Major issues</td>
<td>- Capital costs (for growth)</td>
<td>- Conversion costs (to e-economy)</td>
</tr>
<tr>
<td></td>
<td>- Developing core competency</td>
<td>- Maintaining core competency</td>
</tr>
<tr>
<td></td>
<td>- Motivating staff</td>
<td>- Retaining staff</td>
</tr>
<tr>
<td>Marketplace discriminators</td>
<td>- Lowest possible cost</td>
<td>- Best value for the money</td>
</tr>
<tr>
<td></td>
<td>- Highest possible quality</td>
<td>- Best-of-breed in quality</td>
</tr>
</tbody>
</table>

Table 4.4: Capitalization Costs for Common IT Jobs

<table>
<thead>
<tr>
<th>Job Category</th>
<th>Cost/Seat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer technician</td>
<td>$12,000</td>
</tr>
<tr>
<td>Digital designer</td>
<td>$35,000</td>
</tr>
<tr>
<td>Manager</td>
<td>$11,000</td>
</tr>
<tr>
<td>Secretary</td>
<td>$9,000</td>
</tr>
<tr>
<td>Software engineer</td>
<td>$24,000</td>
</tr>
<tr>
<td>System engineer</td>
<td>$20,000</td>
</tr>
</tbody>
</table>
expressed in dollars/seat capitalization cost, should be for representative information technology jobs. Software engineering needs seem quite modest compared to digital designers. The requirement for laboratory space and expensive computer-aided design (CAD) tools increase the capital costs for digital designers. Managers and secretaries have very basic needs. Thus, their capitalization costs are relatively low.

DEPRECIATION CONCEPTS

It is important to note that capitalized items held for more than a year are normally depreciated. Depreciation is a system of accounting that lets organizations distribute the costs of a tangible asset, less its salvage value (if any), over its estimated useful life. From a tax point of view, depreciation lets the firm write off part of the cost of an asset for obsolescence, exhaustion, and wear and tear so that it can be replaced without undue tax consequences. Normally, the value or basis of the asset is its cost. The useful life of the asset is its expected service time. If the asset is held for more than one year, it is treated as a capital expenditure. The estimated salvage value is the amount that is expected to be realized when the asset is retired from service.

There are several commonly accepted depreciation methods. The method most often used is called straight-line depreciation. Under the straight-line method, the salvage value is first subtracted from the original cost. Then the result is divided by the useful life to compute the annual depreciation charge. For example, you would be able to write off $2,000 a year against depreciation if you capitalized a $10,000 server whose useful life is five years, assuming that there is no salvage value on retirement. Depreciation lets you write off your investment over several years. On the other hand, you can expense $20,000 in any single year with the United States if this is more beneficial.

From a tax point of view, it is to your benefit to depreciate the asset as quickly as possible. Methods such as the declining balance and sum of the years digits support taking larger amounts of depreciation earlier than the straight-line method. To prevent too much from being depreciated too quickly, the Internal Revenue Service (IRS) has established rigid accounting guidelines for computer equipment and software. The IRS dictates what depreciation method can be used and what the useful life is. For example, computers are normally depreciated over a period of five years using the straight-line method under the assumption of no salvage value. Printers are written off over a period of three
years. To comply with current tax laws, you should consult your financial people or your accountant to get help with the calculations and the most current information on the tax codes.

Trade studies are often required when making capital decisions. Selection of equipment and software involves alternatives, and management will want you to justify your selections. They will also want to know the criteria you used for your analysis. Besides doing a complete and technically competent evaluation, they will want to make sure that you were fair and impartial in your judgments. One of the more critical trade studies (a focus in Chapter 6) is called make/buy analysis. Make/buy analysis answers questions such as these: Does your staff develop the product or do you employ a commercial product? Do you perform the function in house or do you hire an outside contractor to do the job?

QUICK-TO-MARKET STRATEGIES

Most commercial firms I have worked with are more interested in shortening their time to market than in cutting costs or increasing productivity. They want to rapidly create and build their business as new markets open and become viable. They want to bring their products to market ahead of their competitors. Time means money in such situations. But the time and effort required to generate the product in staff-months of labor aren’t always interchangeable. As Fred Brooks said so aptly in his classic The Mythical Man-Month, “adding manpower to a late project often makes it later,” and “just because a woman can have a baby in nine months doesn’t mean nine women working together can have a baby in one month” [Brooks, 1995]. There are some underlying laws of nature that money can’t change.

To speed the process, many organizations use rapid application development (RAD) and other lightweight methods like extreme programming [Beck, 1999]. They break the job into its parts and schedule the work so that it can be done iteratively, in parallel, or in spurts. They then streamline the process and improve efficiency by doing things such as eliminating tasks, reducing the time per task, reducing backtracking, and increasing the effective workweek. In pursuit of these strategies, they try to take advantage of the approaches that appear in the RAD opportunity tree illustrated in Figure 4.4 [Boehm, 2000].

Justifying ideas or improvements using quick-to-market strategies is difficult because comparative time-line benchmarks don’t exist in most industries. How can you demonstrate increased speed and agility if you don’t know how
Figure 4.4: RAD Opportunity Tree


long it should take your staff to bring a product to market? You might try time lines for similar projects, but there may have been a lot of noise in the past performance data. For example, a project may have taken a year to bring a product to market. The forcing function for this schedule was an industry show that represented a major market window. If you didn’t make this show, you missed your
product launch opportunity because this is the only time all your customers gather in one place to view new product offerings (e.g., a consumer electronics show). In the rush to make the show, many compromises were made. As a result, the product was neither fully functional nor of the quality expected. While it looked good, performance was also slower than expected. Consequently, using such a project as a benchmark for other projects would be a major mistake.

I recommend using calibrated cost models to generate benchmarks for comparison. Calibrated cost models can generate duration estimates that are within 20 percent of your actual experience 80 percent of the time when they are properly calibrated [Boehm, 2000]. Of course, calibration is not an easy task in most organizations. However, the credibility it adds is worth the effort.

Guidelines of what to do and what not to do when using time to market as the basis of justification for your improvements are listed in Table 4.5. They

<table>
<thead>
<tr>
<th>What to Do</th>
<th>What Not to Do</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ As your first option, be prepared to streamline the work and plan your work tasks so they can be done in parallel.</td>
<td>▪ Don’t promise the impossible; relate your schedules to desired capabilities instead of delivery dates set in stone.</td>
</tr>
<tr>
<td>▪ Determine a realistic schedule before you negotiate your timeline for delivery.</td>
<td>▪ Don’t assume that the past is predictive of the future; there are often biases in the data that make it unreliable in establishing benchmarks.</td>
</tr>
<tr>
<td>▪ Increase management’s flexibility by keeping profit-and-loss responsibility as low as possible in your organization.</td>
<td>▪ Don’t schedule a death march project [Yourdon, 1997]; people burn out and leave when confronted with unreasonable expectations and continuous adversity.</td>
</tr>
<tr>
<td>▪ Avoid backtracking and false starts; these two culprits are a major cause of schedule delays.</td>
<td>▪ Don’t streamline tasks on the critical path without fully analyzing the impacts.</td>
</tr>
<tr>
<td>▪ Use calibrated software cost models to generate comparison benchmarks.</td>
<td>▪ Don’t forget to seek major improvements via simple changes.</td>
</tr>
<tr>
<td>▪ Recognize the biases associated with your past performance data.</td>
<td>▪ Don’t rely only on efficiencies to achieve desired improvements; corporate culture and personal preferences need to be taken into account as well.</td>
</tr>
<tr>
<td>▪ Minimize the bureaucracy to maximize the organizational efficiency.</td>
<td></td>
</tr>
<tr>
<td>▪ Manage the risks associated with your actions to increase the probability of success.</td>
<td></td>
</tr>
</tbody>
</table>
assume that you are pursuing one of the strategies listed in the RAD Opportunity Tree in Figure 4.4. For example, you could speed things up by doing noncritical path tasks in parallel. Although this will increase your risk, you may be willing to tolerate it because it makes shorter schedules achievable. As a preventive measure, you decide to put proven risk management techniques to work on the project to improve your chances for success [Reifer, 1997a]. These techniques focus your attention on tracking and controlling the many variables that can adversely affect schedule performance (vacation schedules, task startup delays, and so on).

ARCHITECTING PRODUCTS USING TIME TO MARKET AS JUSTIFICATION

In Chapter 7, I present a case study that justifies the move to product lines, architectures, and systematic software reuse using shortened time to market. Many firms are adopting architecture-centric strategies because, when adequately populated with components, they reduce the amount of work that is required to get a product out the door. The technology, best processes, and experience associated with software reuse are proven, available, and well documented in the open literature (see [McClure, 1997; Reifer, 1997]). As Figure 4.4 shows, architecture-based software reuse allows you to cut time to market by eliminating tasks and making the job easier.

Moving to product line management and architectural concepts is an extremely difficult task to accomplish in project-based organizations. As I stated in Chapter 1, the primary reason for these difficulties is that it is hard to share assets across projects because of the way work is budgeted and people are rewarded. In project-based organizations, software engineers and managers are rewarded for their contributions to the project, not a product line. “Are the benefits worth the effort?” you are probably asking. Most experience reports on the topic say, “yes” [Lim, 1998]. The organizations involved have reported substantial savings in time and effort with architecture-based software reuse when these barriers have been overcome [Jacobson, 1997]. In addition, software defect rates have been reduced by a factor of up to 10 compared to industrywide benchmarks when reuse technology has been brought into use [Reifer, 1997b].

To take full advantage of product line and architectural concepts, you must make changes in your management infrastructure. Figure 4.5 summarizes the
Defined process  
- Domain engineering (develop and maintain the architecture)
- Application engineering (develop and maintain the product)
- Asset management (build the right assets and make them available to the software developers)

Product line concepts  
- Focus by management on product lines and families, not projects

Known decision framework  
- Policies, practices, guidelines, metrics

Responsive organization  
- Councils, advocacy, ownership, deployment teams

Sharing incentives  
- Bonuses, promotion, prestige, royalties

Distribution mechanisms  
- Library, catalog, newsletter, training

**Figure 4.5: Reuse Infrastructure Needs**

changes needed to put reuse concepts to work. The resulting process (Figure 4.6 [Reifer, 1997b]) is what people in the software reuse business call the "dual life cycle paradigm." Based on this paradigm, the domain engineering process (it includes the activities of domain analysis and implementation) develops/evolves the architecture to serve as a framework for product line and family development. The applications engineering process is pursued in parallel to take advantage of the architecture and develop products that take full advantage of the repository of reusable software assets built to support it. Assets in this sense are any product of the software development life cycle that can be potentially reused (architectures, designs, code, tests, and so on). They include both new and COTS components.

The asset management process is established to manage the assets that are put into the repository so that they can be made available to potential users in an acceptable form. The paradigm is called dual life cycle because the architecture is refined in parallel as applications are developed.

The software reuse library is the mechanism that is used to manage the assets and handle their distribution. The beauty of this approach is that specialists can perform architecture activities without adversely affecting the applications
engineering process. In addition, each process can select the development approach it believes is best for the situation. For example, architecture could be developed iteratively as applications are developed using any modern approach.

This paradigm puts a premium on architecture development. For the purposes of our discussion, architecture is defined as a structure describing the components, their interrelationships, and the principles and guidelines governing their design and evolution over time [Bass, 1998].

The following rules of thumb are extracted from my own and others’ experience [Poulin, 1997]. They should help you estimate the effort and duration of a reuse program structured using an architectural basis and the dual life cycle paradigm.
20/80 rule  Twenty percent of your assets will be responsible for 80 percent of the reuse; 20 percent of your users will be responsible for 80 percent of the results.

Design for reuse rule  The additional cost to develop an asset for reuse will range from 35 to 50 percent of nominal costs.

Design with reuse rule  The cost with reuse averages between 20 to 25 percent of nominal costs when the component was designed with reuse in mind.

Three strikes you're out rule  The breakeven point for developing an asset to be reusable is three instances of use. In other words, don't develop an asset to be reusable unless three or more projects within the product line or family agree to use it within a specified time period.

Ten times the norm in quality rule  Because of their extensive use, expect reusable assets to exhibit ten times the quality when compared to the norm.

Throwaway code rule  Expect less than 10 percent of the code that is advertised as throwaway code to be thrown away.

Domain engineering rule of six  It takes six experienced engineers an average six months to do a full domain analysis and a year to develop a reference architecture.

I share these rules of thumb with you because I use some of them in the case study in Chapter 7. If you are interested in additional references on product line management and architecture concepts, Appendix A provides annotated descriptions of several books on this topic that I feel are worth reading.

MAKE-VERSUS-BUY ANALYSIS

One of the key considerations covered in the case study in Chapter 7 is whether to make or buy the software. To make the software, you would deploy the resources (skilled people, equipment, facilities, tools, infrastructure, and so on) that are available in house to get the job done. If these resources weren't available, you might consider contracting for or outsourcing the work to qualified sources. As an option, you could obtain needed resources by hiring more permanent or part-time staff or by making selected acquisitions of firms with needed capabilities. In either case, you have to make sure that your future workload could sustain additional employees because you don't want to hire and fire them at the whim of fluctuations in the workload.
Table 4.6: Make versus Buy: Difference in Mind-sets

<table>
<thead>
<tr>
<th>In-house Software</th>
<th>COTS Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developed to satisfy requirements.</td>
<td>Developed to satisfy market need or niche.</td>
</tr>
<tr>
<td>Engineered for development.</td>
<td>Engineered for evolution.</td>
</tr>
<tr>
<td>Internal forces drive content.</td>
<td>Market drives content.</td>
</tr>
<tr>
<td>Integration is a natural part of the overall development process.</td>
<td>Integration is often difficult; addressed via glue code and wrappers.</td>
</tr>
<tr>
<td>Maintenance is where money is made.</td>
<td>Maintenance is a nightmare for customer (new versions must be considered and evaluated).</td>
</tr>
<tr>
<td>Primary asset is the component design.</td>
<td>Primary asset is the product architecture.</td>
</tr>
</tbody>
</table>

The buy option should also consider the use of commercial off-the-shelf software. COTS software with most of the desired functionality is provided by some third party. It has the advantage that you don’t have to maintain the software once it is deployed; the firm providing the software takes on this responsibility. While seemingly attractive, use of COTS software is risky because others are in control of its evolution. To take full advantage of COTS, you need to understand that it was developed with different goals in mind. See Table 4.6 for a summary of the differences between COTS and software developed in house.

To evaluate the financial tradeoffs involved in using COTS, you must consider the differences given in the table as you evaluate the alternatives. For example, many firms contrast the COTS licensing costs directly with their software development effort. For the comparison to be fair, the cost tradeoffs should consider the glue code or wrapper development effort. The comparisons also need to take operations and maintenance costs into account. For custom software, sustaining costs can be substantial, especially if the software is released with lots of bugs. In the case of COTS, maintenance involves license renewals and glue code repair.

**MOVING TO A WEB-BASED ECONOMY**

In Chapter 8, I present a case study that puts a value on an Internet startup firm you are considering acquiring. This kind of investment decision is difficult to make because you have to assess the worth of the firm’s intellectual property.
Dealing with startups is extremely complicated, especially when you must deal with legal challenges inherent in a networked economy, determine the value of an Internet startup, and put a value on knowledge as capital.

We all know and use the Internet and the World Wide Web (WWW). Put in its most basic form, the Internet is a collection of interconnected networks of computers that communicate throughout the world via packet-switching protocols. The Web uses the Internet to provide its users with links to more than 100,000 networks, consisting of over 10 million host computers located in over 100 nations. The Web and the Internet have revolutionized business and created a new economy, new tools, and new rules. There are lots of money, people, and businesses on the Web. To thrive in the e-commerce world, organizations are changing their management infrastructures and adjusting the way they do business. They are learning to manage the flow of information and employ knowledge as capital as they throw away their often outdated business practices. They are being taught that they must be able to react quickly, innovate ceaselessly, pursue alliances, and handle change continually.

**Capital Is Abundant**

People are moving to e-business because that’s where the markets are, and fortunes are being made and lost. Even though the failure rates for Web startup firms is high, more initial public offerings (IPOs) seem to be coming to market every day. Many liken today’s e-business and e-commerce world to the days of the Gold Rush. Few prospectors actually got rich quick by striking pay dirt. Most of those who made fortunes did so by providing the prospectors with the tools, mules, and supplies they needed to start up. I believe that the people who provision firms making a transition with tools, training, and support will prosper as business on the Web expands.

As shown in Figure 4.7, the venture capital used to fuel new developments along these lines is still plentiful according to the U.S. Department of Commerce [DoC, 2000]. The challenge is bringing the idea to market first.

**Concerns Associated with Electronic Commerce**

The Internet also provides a conduit for many types of commercial and financial transactions. Consumers are utilizing on-line banking, brokerage, real estate, shopping, and countless other services. Business-to-business (B-to-B) sites have emerged whose primary function is to promote on-line bartering with suppliers
and create virtual warehouses. Contracts for goods and services are made online without face-to-face meetings of participants. Most important, e-commerce is changing the manner in which firms use the Web. For example, General Electric and many other large organizations are cutting costs and improving customer satisfaction via the Web. They are using its capabilities to coordinate purchases, lever buying power, and tap on-line markets for goods and services.

In the case study in Chapter 8, a large firm decides to acquire a smaller Internet startup to secure e-business and e-commerce skills. The challenge the larger firm faces is deciding how much to pay for the smaller firm, whose only assets are people and intellectual capital. The case develops a unique framework for making such a determination. It builds on concepts taken from the knowledge management field to place a value on the intellectual capital of the Internet startup [Roberts, 2001].

**Determining Value of Startups**

In determining the value of startups, you must remember that knowledge by itself has no value. Knowledge by its very nature takes on value only when people use it to solve a problem. Therefore, as shown in Figure 4.8, for an organization to have value, its people, markets, and intellectual property must provide recognized expertise in a specialized domain. In other words, the startup must have the
human skills, intuition and wisdom to bring the right products (and services) to market before the competition.

How would you put a value on an Internet startup whose only assets were people, ideas, and venture funding? To assess the startup’s investment potential, you would start by evaluating the factors that influence its ability to make a profit (many of these factors are listed in the business plan outline in Chapter 2). As you will see in Chapter 8, we use such traditional models to determine the return on investment and breakeven point associated with a startup. Many Internet startups don’t make profits for years, but their principals can get rich if they catch the imagination of the investing public during their initial stock offering.

The more challenging exercise is determining a value for the firm based on the notions that software is knowledge and knowledge is capital. For the purpose of our discussion, “knowledge capital” refers to the ideas that facilitate production of goods and services. To put a cap on their investment, potential backers want to know how much the “knowledge capital” the firm owns is worth. Making the appraisal is especially difficult because bright people can generate great products that don’t sell. This forces us to separate the ability to innovate from the facility to sell.

Although I don’t get into appraising the worth of “knowledge capital” in Chapter 8, I share with you a framework I have developed for this purpose. This structure is displayed in Figure 4.9 and amplified briefly here:

- Recognition that people are the primary resource, not tools, equipment, and machinery
- Emphasis on investing in getting smarter, not stronger
Knowledge Capital

Elements
- Ideas
- Skills
- Ability to innovate
- Nimbleness
- Know-how

Measures
- Number of new products generated
- Accumulated years of experience and certifications
- Apparent productivity
- Response time to changes
- Ability to use skills to innovate and generate products

Figure 4.9: Framework for Assessing Knowledge Capital

- Focus on developing both the learning and the know-how needed to use available skills
- Realization that responding to changing markets is as important as being innovative
- Putting a premium on generating new products, not on developing new ideas

While all the factors in this framework are quantitative, most are hard to measure. In response, I have developed a more qualitative evaluation scheme that I use in Chapter 8.

If you are interested in a good book on this topic, read the delightful text by Sveiby and Lloyd about adding value by valuing creativity [Sveiby, 1987]. Most of the other books I have read on the topic of innovation discuss process, not measurement, because even though it is easy to measure the effect of innovation after the fact, it is hard to establish a value for it when it appears before your eyes. That’s the purpose of the framework illustrated in Figure 4.9.

**SUMMARY**

This chapter serves as an introduction to the case studies that follow in Chapters 5 through 8. It summarizes the case studies and highlights how materials covered in the earlier chapters of the book will be used to build a business case. Besides
providing necessary background information, the chapter prepares you to delve in the world of numbers.

**Observations**

The key point of this chapter is that it is almost impossible to separate software from organizational improvement activities. Factors that influence improvement in one domain impact the other. That’s because few organizations generate software in a vacuum. Their goal is to use software to make a profit. For example, software in cellular phones provides functions and features that provide marketplace discrimination. As another example, packaged software sells well when the organizations buying it believe they are getting good value for their money and customer support will be above average. In response, software engineers need to understand how to relate what they are trying to accomplish to the broader organizational improvement goals. From my experience, I believe they will be better able to acquire senior management support when they are armed for the battle of the budget with sound business as well as technical cases. The case studies in Chapters 5 through 8 provide you with examples of how to accomplish this feat.

**KEY POINTS**

- Like the Chinese emperor, you can improve productivity by taking advantage of the dual nature of software productivity.
- Whenever making cost or productivity comparisons, develop industry benchmarks to serve as measurement yardsticks.
- Process improvement can cut costs by an estimated 11 to 15 percent when an organization jumps a level in the Software Engineering Institute’s capability maturity model.
- Cost avoidance is preferred to cost reduction because it deals with future expenditures.
- Capitalizing software provides good people with the equipment, facilities, and tools they need to get the job done efficiently.
- Reductions in time-to-market strategies can be accomplished by streamlining the work, reducing backtracking, and performing tasks in parallel.
- Migration to the use of product line, architecture, and systematic reuse concepts requires major investments and infrastructure changes to facilitate sharing.
✓ The dual life cycle paradigm focuses on managing architecture and development tasks as they are conducted in parallel.
✓ Moving to a Web-based economy presents the community with many challenges.
✓ Determining the value of Internet startups requires us to develop new models and metrics.

References


