

## **Chapter 7**

# **Acquisition Planning**

---

## Contents

<b>7.1 Planning is Key to Success</b> .....	7-4
<b>7.2 Strategic Planning Goals</b> .....	7-5
7.2.1 Program Stability .....	7-7
7.2.2 Quality .....	7-7
7.2.3 Supportability .....	7-8
7.2.4 Cost and Schedule .....	7-8
<b>7.3 Software Acquisition Strategy</b> .....	7-10
7.3.1 Mission Definition .....	7-11
7.3.2 Acquisition Strategy Development .....	7-11
7.3.2.1 Competition .....	7-12
7.3.2.2 Concurrency/Time Phasing .....	7-13
7.3.2.3 Design-to-Cost .....	7-14
7.3.2.4 Performance Demonstrations .....	7-15
7.3.2.5 Performance Incentives .....	7-15
7.3.2.6 Make-or-Buy .....	7-16
7.3.2.7 Pre-planned Product Improvement (P3I) .....	7-16
<b>7.4 Program Planning Process</b> .....	7-17
7.4.1 Planning Objectives .....	7-18
7.4.2 Planning Scope .....	7-19
7.4.3 Recommendations for Program Planners .....	7-19
<b>7.5 Program Decomposition</b> .....	7-20
7.5.1 System/Segment Specification (SSS) .....	7-20
7.5.2 Work Breakdown Structure (WBS) .....	7-21
7.5.2.1 WBS Interrelationships .....	7-21
7.5.2.2 Prime Mission Product Summary WBS .....	7-22
7.5.2.3 Software Project Summary WBS .....	7-22
7.5.2.4 Software Contract WBS .....	7-24
7.5.2.5 Software Project WBS .....	7-26
<b>7.6 Market Analysis</b> .....	7-27
7.6.1 Software Product Definition and Decomposition .....	7-27
<b>7.7 Baseline Estimates</b> .....	7-28
7.7.1 Estimation Accuracy .....	7-30
7.7.1.1 Program Estimate Selection .....	7-31
<b>7.8 Continuous Program Planning</b> .....	7-31
7.8.1 Continuous Planning Recommendations .....	7-33

**7.9 Other Planning Considerations** ..... 7-34

    7.9.1 Major Milestones and Baselines ..... 7-34

    7.9.2 Program Budgeting and Funding ..... 7-36

**7.10 References** ..... 7-37

---

## 7.1 Planning is Key to Success

An analogy can be made between planning for the acquisition and management of major software-intensive systems and planning for a military campaign. Planning for combat was explained by General H. Norman Schwarzkopf.

*“I want to emphasize the importance of focusing on the enemy when planning and conducting combat operations. First, you must know your enemy. Second, you must develop your plan keeping the enemy foremost in mind. Third, you must wargame your plan to enhance your ability to develop or adjust the plan once enemy contact is made.” [SCHWARZKOPF88]*

You must view cost overruns, schedule slips, and performance shortfalls as your enemy. You must also emphasize the importance of focusing on risk when planning for and conducting your program. First, you must know your program-specific risks. Second, you must formulate your strategy to enhance your ability to develop or adjust your plan as you encounter new sources of risk in the ever-changing program environment.

*World-class software doesn't just happen — it's planned!* Planning is the most pivotal activity you will perform as a program manager. Planning, combined with process improvement, is a continuous activity that must be revisited and improved upon throughout the life of a software-intensive system. A poorly planned software program is one that is doomed to failure. Through proper and careful planning you can address and deal with the five critical factors that determine the success or failure of a software program:

- Quality
- Cost
- Schedule
- Performance
- Supportability

Although software planning is performed throughout the software life cycle, *up-front, strategic planning* is the most crucial. It addresses these critical planning factors that get exponentially more costly to deal with in later phases. Software development is not an exact science, but using a combination of good historical data and systematic techniques can improve the accuracy of your estimations. The F-22 program illustrates that software development success is achievable through careful risk management and knowledgeable strategic program planning that combines a keen sense of lessons-learned with a commitment to achieve insightful, intelligent, and creative process improvement. It involves interaction with Air Force agencies and strategic planning stakeholders to arrive at the best software solution within budgeted resources.

In April of 1991, General Merrill A. McPeak, Air Force Chief of Staff, was proud to announce the winner of the advanced tactical fighter (ATF) air superiority aircraft competition and the Air Force's new *Top Gun*. The contract award for the future, fast, agile, stealthy super cruiser, the F-22, was the result of a 54-month Demonstration /Validation (Dem/Val), where two contractor teams dueled for Air Force favor in an unprecedented, risk reducing, joint government/industry-sponsored face-off. [EASTERBROOK92] Applying lessons-learned from the B-2, Air Force planners had the more complex ATF fighter airborne in just four years. For the F-22 software

development effort, strategic planning made the difference. Our ATF planners must be commended on their success. Former Secretary of the Air Force, Donald B. Rice, praised them when addressing the House Armed Services Committee after the award of the F-22 contract. He stated that there has never been a defense program “*that invests as much in the front end...and is in as confident a position to enter full-scale development as the ATF.*” [RICE91] The critical nature of software in the weapons and information systems you are developing or maintaining today, mandates that you take every action necessary to ensure program success.

When trying to assure the overall success of your program, there are three important points to remember when planning your software acquisition.

1. **Software is always on the critical path.** It is usually the biggest cost item in major DoD software-intensive acquisitions, and is also the *highest risk item that must be steadfastly managed*. The ATF planners used a strategy that included lessons-learned from the C-17 development where the software element was considered an unrealistically low-risk item. [REILY92] To better manage the F-22 software development, they decided that software costs must be tracked separately from hardware costs in the Engineering Manufacturing Development (EMD) phase. This was the first time the Air Force had taken this approach to manage and reduce the risk of software cost escalations on a new aircraft. [HUGHES92]
2. **Strive for consistency and completeness.** Consistency means having single standard languages (i.e., Ada) where possible, a standard terminology, a standard software engineering environment (SEE) used by all subcontractor team members, and a strong configuration management program. Completeness means, quite simply, *good documentation*.

**CAUTION: In planning for “good documentation,” do not fall into the “excessive” documentation trap! Less required documentation (i.e., only that which is necessary for technical software development and maintenance) of higher quality is the goal.**

3. **Keep government personnel abreast of the development process** so they know what is going on and understand the software and how it works. An important ingredient in program success is to be an enlightened and supportive customer. If the contractor encounters unforeseen problems, do not criticize and throw rocks — but cooperate in the search for a solution.

## 7.2 Strategic Planning Goals

At the Pentagon there is a sign posted above the DoD Joint Staff office door (quoted from Field Marshal Helmuth Graf von Moltke) that says:

*“Planning is everything. Plans are nothing.”*

The meaning behind von Moltke’s statement lies in understanding the concept of a process-focused approach to problem solving, as opposed to a *product-focused approach*. Planning success is achieved though the planning process.

In 1988-89, the Defense Systems Management College (DSMC) and Harvard University analyzed the findings of several commissions held over a twelve year period and their own extended

research to determine why there were so many defense acquisition program failures. The objective of the study was to determine how Government might learn to “do business like business.” [DSMC89] This theme was reiterated more recently by John Deutch, Under Secretary of Defense for acquisition and technology when he said:

*We have built up a separate way of doing business with DoD that is entirely different than... the commercial sector. There will be a time when a separate defense industrial complex will become too costly to maintain. We have to learn to rely much more strongly on doing business like the private sector.* [DEUTCH93]

The strategic planning goals leading to program success are listed below and are also illustrated in Figure 7-1.

- Program stability,
- Quality (including performance),
- Supportability, and
- Cost and schedule.

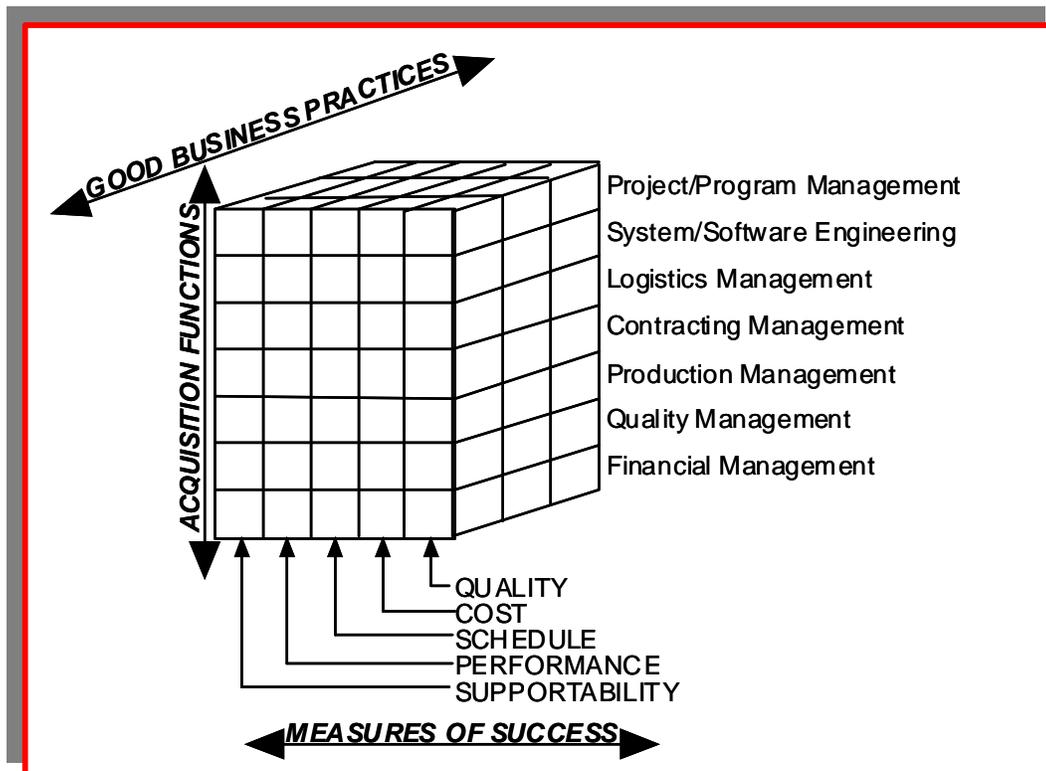


Figure 7-1. Strategic Planning Goals for Program Success

Achieving stability, quality, supportability, schedule, and budget are all determinants of software program success. A realistic cost and schedule that enables built-in product quality enforces on-time completion, which enhances program stability. Program stability, cost, and schedule are always joined at the hip. A stable program can be executed more quickly than one, which is constantly changing, or subject to change in an unforeseen way. Also, a program completed within its projected schedule is subject to the forces of change for the minimum time possible.

Poor quality can cause schedule overruns leading to cost overruns. This can be particularly true where the true quality is not known until final testing is complete and/or the software is in the user's hands. Post-deployment might be the first time you learn that your software simply does not work. Lessons-learned from past DoD failures to meet schedule objectives show that *schedule slips promote excessive changes in requirements*. Users having to wait inordinately long periods before their needs are satisfied, invariably identify additional requirements as time and technology advance.

---

## 7.2.1 Program Stability

*“Recognition of the distinction between a stable system and an unstable one is vital for management . . . A stable system is one whose performance is predictable. It is reached by removal, one by one, of special causes of trouble, best detected by statistical signal.”* — W. Edwards Deming [DEMING86]

The DoD acquisition process has environmental factors not found in the commercial world, such as congressional oversight with one year funding constraints on how and where Defense dollars are spent. Given these differences, it is uniformly acknowledged that program stability is the one business practice that should be institutionalized in DoD acquisition policy. It is also the first or primary goal you should strive to achieve in your program planning efforts, which should ripple across all traditional acquisition functions (e.g., engineering, logistics, and financial management). The key attributes of program stability are *steadiness of purpose*, a *firmly established plan*, and a *supportive system*. Your strategic planning process must link program objectives to resources (time, people, funds, and technology). [Resource estimation is discussed in Chapter 13, *Software Estimation, Measurement, and Metrics*.] It must organize these resources and define a process for achieving the approval of all stakeholders to guarantee the implementation of your strategic plan. It must then guide the development phase and provide for the integration of the effort. Your approved plan should be a product of systematic consensus and a clear decision process.

Maintaining stability in a program that must be accelerated in order to meet a military threat, or that has had its budget cut by 50%, is often a significant management challenge. However, the steps you take to achieve stability (i.e., having an established plan and understanding how your resources are tied to program objectives) can help you restabilize after a change. Not having a flexible plan that adapts to change will often mean chaos added to chaos when your budget gets cut or requirements are added or modified. The time you spend defining your acquisition strategy early on will go a long way in assuring stability throughout the entire life of the system, especially during the critical acquisition years. The Cost Analysis Requirements Document (CARD) [discussed below] is an excellent tool for structuring your program for stability.

---

## 7.2.2 Quality

Without exception, the second most important goal must be product quality. In the past, the goals of cost and schedule often took priority over quality because they had the highest visibility during early phases of development. Until the testing phase, quality is essentially an *unknown* or *invisible commodity*. The rewards for software with good operations and maintenance records are usually not enjoyed by the program manager who developed and delivered the product. This

lack of positive feedback creates managers who let costs and schedules drive their decisions — often at the expense of quality. This is “*penny-wise and pound foolish!*”

If you let cost and schedule take priority over quality, by the time your software is coded and ready for testing, it can be so riddled with defects that dynamic testing is painfully time-consuming, costly, and difficult. Once the software is in the user’s hands, poor quality becomes excruciatingly visible because the cost to fix garbage code is exponentially greater than the cost of building a quality product (not to mention the damage to user confidence poor quality causes). [KINDL92] With quality as a key planning goal, you will produce a *good product* on a *predictable schedule* at a *predictable cost* with the *desired performance*. Remember, one of the most important ingredients in producing quality software is the belief in the importance of its mission and an associated *commitment* at all levels to support that belief. *Remember, the true success of your software can only be determined from a life cycle perspective.* To deliver a quality product you must be willing to adjust cost, schedule, and resource allocation to support the quality goal, however you define that goal.

Sufficient performance, a quality attribute, is defined in terms of mission capability, supportability, life cycle costs, and unit costs. Beware, rigid or excessive system performance requirements can drive costs unnecessarily high and stretch out schedules. The metrics you use to define your performance goals will ultimately help to determine quality and cost. [See Chapter 13, *Software Estimation, Measurement, and Metrics* for a discussion on these metrics.] Pre-planned product improvement (P3I) and evolutionary development are the standard approaches used to obtain desired technology or features not available at planned schedule cutoff points and milestone decisions. An iterative, evolutionary design process allows for flexible development that advantageously considers performance tradeoffs as the design evolves.

---

### 7.2.3 Supportability

A system that cannot be supported after it is built quickly becomes useless. Worse, when a system is found to be unsupported, previously used systems may have already been dismantled or removed. Too often, a project has been completed, declared a success, and its builders given accolades for their performance, only to be followed by the discovery in the field that it cannot be maintained. Supportability must be planned and designed into the system from the beginning. Trying to add it later can only be done at great cost, if at all. Ensure that your planning includes supportability as a strategic goal.

---

### 7.2.4 Cost and Schedule

The successful F-22 acquisition strategy set a precedent for holding down the price of future DoD weapons systems. Because contractors were required to gamble their own funds, there was great incentive to propose cost-effective solutions. One competing program manager remarked,

*“Except for the investment, Dem/Val was great. We probably developed the technology in half the time we would have if we had not had a competition and a good, big team.”* — Thomas R. Rooney [ROONEY90]

Throughout this chapter the words “cost” and “schedule” are used over and over because they are two critical metrics used to assess program performance. In DoD, they are often the drivers used to define program management practices. Their importance is not surprising given the pressing need to update old systems and develop new ones to meet new requirements or threats in today’s resource-constrained Defense environment. As General John W. Vessey Jr., while Chairman of the Joint Chiefs of Staff, explained,

*“Resource-constrained” environment [are] fancy Pentagon words that mean there isn’t enough money to go around.” [VESSEY84]*

Staying within budget has been one of the most difficult software management goals to achieve. Much of the difficulty arises from the DoD budgeting and funding process. Long lead times are needed to get money committed; therefore, program costs must be projected long before software requirements are defined and software cost elements can be realistically estimated. Unfortunately, systems planners are impacted when insufficient dollars are allocated to the software element (on the system’s critical path) which in turn often causes the overall program schedule to slip. [MARCINIAK90]

The RFP should require that offerors provide a development schedule appropriate to the known requirements, showing all major milestones, audits, reviews, inspections, and deliverables. It is expected that this schedule will change as requirements become better defined. You must evaluate this schedule to determine if the offeror understands the need for presenting detailed schedule information and for tying that information to detailed program task requirements. You must also determine whether the program tracking system being proposed is part of the company’s normal management practices or if it is new for this program. Also, you will want to ensure schedule needs and types are described and included in the Software Development Plan (SDP).

Problems are often created when schedule baselines are established before software requirements are well defined and understood. Government RFP preparers may include schedule information based on factors that do not take into account the system development process or software requirements. Offerors then inadvertently accept RFP schedule information as a requirement for a responsive proposal, and prepare their response based on these *so-called* requirements. This practice causes offerors to bid to untenable schedules affecting the viability of their submissions, decreasing the probability they will complete tasks as proposed. One solution is to provide minimum schedule guidance, and to require that offerors propose development schedules based on program requirements and their own development approach.

Where users remain adamant that arbitrary delivery dates must be met, you will do well to work with them on the concept of evolutionary and/or incremental deliveries versus a full scope capability. Even then, it is recommended that you use every persuasive power at your command to educate them on the exceedingly high failure rate for programs with unrealistic schedules.

**NOTE: DoD is more interested in receiving a quality product on a predictable schedule at a predictable cost than in setting arbitrary target dates which may not be achievable.**

---

## 7.3 Software Acquisition Strategy

Acquisition strategy has been defined as a *master plan, a road map, a blue print, and a plan-to-plan-by* to achieve program goals and objectives. Every major software-intensive development has the possibility of failure. Your acquisition strategy serves as a means for reducing the odds of program defeat through the organized preparation of a plan to minimize software risk. It serves as a guide to direct and control the program, and as a framework to integrate those functional activities essential to fielding a totally operational system — not just pieces of hardware and software. The conceptual basis of the overall plan — the *objective* — is what you must follow during program execution. It also serves as the basis for all program management documents, such as the Acquisition Plan, the Test and Evaluation Master Plan (TEMP), the Integrated Logistics Support Plan, and the Systems Engineering Master Plan (SEMP).

As you learned in Chapter 6, *Risk Management*, your Acquisition Plan must address, deal with, and identify risk issues and alternative solutions. You must decide on what type of contracting strategy to employ, such as design-to-cost, award fee/incentives, or to make-or-buy your software element, which brings with it the issue of data rights. Your development methodology might include concurrency or time phasing of development phases, prototyping, P3I, evolutionary acquisition, and/or incremental development. The supportability of your software must also be part of your acquisition strategy which includes the requirement for an open systems architecture. Other alternatives include the design and use of reusable assets; re-engineering as a re-development alternative; assessing your potential suppliers' development maturity; and tracking and controlling risk elements. Additional factors you should address are the cost of scrap and rework, program budgeting and funding risks, a forecast of how future technologies might impact your development, and what kind of planning and management tools you can employ to facilitate your planning activities.

Because acquisition strategies for software systems abound, you should conduct a meaningful lessons-learned exercise before settling on your final acquisition and development plan of action. The ATF engineering and manufacturing development (EMD) approach and contract were strongly influenced by careful review of how a whole procession of prior avionics developments went astray. According to Colonel Borky, ATF planners based their final acquisition strategy on avoiding the following list of *classic* mistakes that get made over and over.

- Unrealistic estimates of time, costs and manpower requirements to execute a development. (Admittedly, we still lack good estimating methods and tools, but there's ample evidence of program managers who willfully understate resource requirements for software to fit within a program budget when they cannot cheat on the hardware estimates.)
- Inadequate planning for software integration and test, including required facilities.
- Allowing contractors to do significant coding before system engineering is complete and requirements are stable. [BORKY91]

Selecting the right type strategy for your program is much the same as selecting the right strategy for winning a battle. When preparing a battle plan, you must first know what you are getting into and what you have to take with you. Sun Tzu, master Chinese strategist and general during the Era of the Warring States (circa 500<sub>BC</sub>), explained:

*“Know the enemy and know yourself; in a hundred battles you will never be in peril... If ignorant both of your enemy and of yourself, you are certain in every battle to be in peril.” [SUN500<sub>BC</sub>]*

Knowing your enemy and yourself, will enable you to understand your mission, identify your enemy, assess the terrain over which you must pass, and from this, determine your tactics. Your tactics, when combined, make up your overall acquisition strategy.

---

### 7.3.1 Mission Definition

Your mission is to deliver a software system that fulfills user requirements, on time, within budget. In many cases, however, the user may not understand exactly what those requirements are, so clarification of the mission may fall on your shoulders. This mission definition includes timing (schedule) and cost constraints that may have an effect on perceived requirements. The enemy can be equated to all those risk factors that conspire against the completion of your mission.

The enemy to a successful acquisition is not always as clearly defined as is the enemy in a conflict between nations. The enemy may include a variety of programmatic risks or constraints such as a short schedule, a limited budget, factors in the development environment (software being developed concurrently with hardware, or software pushing the leading edge), or the requirement to build software designed for reuse and easy maintenance. Assessing the terrain involves analyzing the business base, the capabilities of the development team, development system hardware, and host system hardware for which the software is being built.

---

### 7.3.2 Acquisition Strategy Development

When preparing your acquisition strategy, you must consider *all* the factors pertaining to the requirement such as:

- Budget
- Technical aspects of the software
- Hardware on which the software will operate
- Obtainability of existing software to satisfy the requirement
- Availability and past performance of organic or contractor developers
- Timing of hardware and software development
- Size and functions required for the program office
- Software reusability
- System maintainability
- *Ad infinitum*

Obviously the list of issues that can affect the software development strategy is long and complicated and will differ among programs. The bottom line is, you must consider every possibility when selecting contractual, schedule, and budgeting tactics in developing your acquisition strategy. It must be derived from a commitment to encircle, outflank, out think, and triumph over the enemy at every encounter. Your strategy must not be a rigid formula, but a flexible framework that can be applied artfully as circumstances dictate. It must also be based on

the application of *common sense* and *best practices* found in these Guidelines to address the inevitable problems that emerge in every major software acquisition due to the extreme complexity of the endeavor itself. Sun Tzu expressed this strategy when he proclaimed:

*“When the enemy is at ease, be able to weary him; when well fed, to starve him; when at rest, to make him move. Appear at places at which he must hasten; move swiftly where he does not expect you.”* [SUN500<sub>BC</sub>]

If your software requirements are relatively risk free, you might choose a straightforward approach. If on the other hand, your software is complex and deeply embedded in a hardware system and/or unprecedented, then your acquisition strategy will be quite complex and must involve extensive research into alternatives. For example, avionics software acquisition usually follows airframe development and avionics hardware selection. The airframe developer may then choose to subcontract the development of the very sophisticated software suite needed to mechanize the complex of weapons delivery systems. Planning for this type arrangement will greatly impact your acquisition strategy. The acquisition strategies commonly used for major DoD software-intensive acquisitions generally include the following concepts:

- Competition,
- Concurrency/time phasing,
- Design-to-cost,
- Performance demonstrations,
- Performance incentives,
- Make-or-buy, and
- Pre-planned product Improvement (P3I).

### 7.3.2.1 Competition

The software industry supplies technology for software development. It also supplies engineering, development planning, management, organization, infrastructure, and processes. These elements were created by industry through efforts to capture market share and gain a competitive edge. Although academia helps sow the seeds of research and training, industry competes to apply better ideas to the software engineering task. Competition is vital to enable technology improvement in defense software acquisition. The forces of innovation are unleashed when industry is challenged to compete for the position of the best supplier.

Defense competition can take many forms. In fact, there may even be no competition. For example, a sole source procurement might be selected due to the nature of the product and the availability of the source. A competition can involve two or more companies and may occur during research and development or implementation. Two generic forms of competition are used in military acquisitions:

- **Design competition.** An example of this was provided in the F-22 procurement discussion. Two or more competing teams of companies develop concept and design approaches, one of which is selected for the production contract. The benefits of this type of competition are a clearer understanding of requirements through multiple perspectives, high risk items are identified and resolved in a more thorough manner, budget commitment is deferred until PDR, and the risk of selecting a poorly qualified FSD contractor (organization) is reduced.
- **Production competition.** Two or more companies bid to secure all or part of a production contract. Where more than one company is employed through initial production, risk is further reduced.

In general, you will be discouraged from dual awards for design and/or partial production because this requires additional funding up front. However, experience has shown that the longer two competing contractors are carried, the greater the opportunity for success.

### 7.3.2.2 Concurrency/Time Phasing

Concurrency is a fast track acquisition strategy that involves the overlapping of design, testing, production, and deployment activities. The overlapping and elimination of phases in the acquisition cycle, as well as overlapping or eliminating activities within a phase, are also choices based on the urgency of the need or the maturity of the system. A realistic technology assessment and allowance for critical time duration activities are key in planning a program with a high degree of concurrency between (or within) phases of the acquisition process.

Concurrency is used in response to a need to get a product to the field within a critical time frame. Short acquisition cycles, abbreviated proposals, condensed statements of work, minimal data reporting, use of commercial practices, and fewer reviews are all used to reduce costs and expedite schedules. A classic example where a concurrency strategy was successfully used was on the Thor missile program. The winning contractor's proposal consisted of a mere 20 pages describing how they intended to manage the program. The contract was awarded in December 1955 and the Thor flew successfully 13 months later.

Another example of a successful concurrency strategy was the Single-Stage Rocket Technology (SSRT) program. The contractor used commercial practices wherever possible which included almost one million lines of COTS test software for controlling ground and flight operations. Nearly 70,000 lines of onboard Ada flight control software were generated using a commercially available autocoding technique that cut costs an order of magnitude over conventional coding methods. They also reduced the number of government/contractor program meetings, saving additional cost and manpower. [WORDEN94]

An example of the risk involved in concurrency was the Sergeant York anti-aircraft gun or "DIVAD" for Division Air Defense, one of the most important weapons systems to be canceled while in production. Concurrency was used to cut normal acquisition time (10+ years) in half and save money. This approach featured parallel development by two competing contractors; the use of off-the-shelf components, a *skunk works* approach with thinly staffed government/contractor program offices shielded from outside scrutiny, contractor flexibility in making cost/performance tradeoffs, limited and combined developmental and operational testing, and a concurrent follow-on development and initial production phase. The strategy stressed minimum government oversight during system development and reduced reviews and reporting requirements. Government access to contractor facilities and information was also limited. [GAO86]

**NOTE: Taken from Al Capp's Li'l Abner comic strip, the term "skunk works" was first used by Lockheed on the U-2 and SR-71 programs. It denotes a separate management operation outside the normal acquisition process due to the highly classified nature of the contractor's work.**

By concurrently developing, testing, and making improvements while in production the gun's considerable software integration problems were never ironed out. The procurement was canceled when it failed to perform during follow-on operational test and evaluation (FOT&E) and the Government was stuck with 64 SGT Yorks at ~\$42 million apiece.

The main advantage of concurrency is the achievement of an early operational capability. Another is that design maturity and operational problems surface sooner through earlier testing. Concurrency, however, introduces the substantial risk of performance shortfalls, schedule overruns, and cost growth, especially in complex, unprecedented software-intensive systems.

### 7.3.2.3 Design-to-Cost

Design-to-cost is designing the system to fall within fixed cost and schedule limitations. It is a proven acquisition tool for obtaining lower unit costs. Design-to-cost forces identification of measurable design parameters which can be prioritized and used as targets in managing cost. Since budget and schedule limits are known up front, this approach can result in better requirements definition and increased efficiencies. The disadvantages are that it forces you to commit to a design-to-cost goal before final software requirements are defined. Hence, the need to sell the program may drive design-to-cost goals down to unrealistic levels. Also, since there are no practical ways to validate life cycle cost estimates, the contractor (or the Government) may choose to *down-scope performance requirements* to meet cost goals.

Computer Sciences Corporation (CSC) successfully used a design-to-cost approach to build a document control and tracking system for a pharmaceutical drug application. This was a process improvement initiative to invest in the development of a tool that would reduce the time it takes to get drugs through the testing and regulatory process (8 years) by 50%. This development had to be accomplished within a fixed time, fixed cost, and above a fixed functional baseline.

To accomplish their goal, they used what they called the "*Blue Chevy Policy*." All they needed was basic transportation — not a Mercedes, not a convertible, or a truck. They needed to provide an adequate solution today — not the ultimate solution tomorrow. Their design-to-cost approach relied heavily on hardware and software commercial off-the-shelf (COTS) products with as little custom development as possible. This required a partnership between the developer and the user to accomplish their common goal within mutually agreed upon architectural constraints. This meant the developers had to have direct and continuous access to the user, especially during the proof-of-concept prototyping process.

Training and documentation were developed in parallel with the system. When the program was completed on time, training began the next day. With a good architecture and user-involved prototyping throughout all phases of development, system functions exceeded expectations. They delivered a "*Loaded Blue Chevy*" and were able to grow the company into new areas of technology. [KEMP94]

### 7.3.2.4 Performance Demonstrations

An example of what can go right is the strategy used by F-22 planners to assess performance requirements fulfillment. Using an open market strategy, each ATF competition team built their own demonstration aircraft using their own funds (50% contractor/50% government) and ideas which the Air Force evaluated during the Dem/Val phase. A less desirable alternative would have been a *paper design* — only testable at the end of full-scale development (FSD). [MRAZ91] This strategy was markedly different from that of the B-2 Bomber where the design was frozen at the beginning of Dem/Val. In effect, the demonstration phase, where alternative approaches are explored and bugs are worked out, was virtually skipped. This meant the B-2 was developed on an essentially noncompetitive basis with a cost-plus contract awarded to a single prime contractor eliminating the possibility of achieving a better design solution and reducing cost and schedule.

One reason ATF Dem/Val testing was such a success was because flight test objectives differed radically from traditional military testing. Instead of checking for compliance with a laundry list of requirements, ATF demonstration aircraft were used to show that the competitors had analytically predicted aircraft behavior by spot-checking the performance and technology issues each team thought were critical. “*We didn’t come into the Dem/Val with what the military thinks of as requirements,*” remarked Lt. Col. W. Jay Jabour, director of the ATF combined test force. “*We said to the contractors: ‘Demonstrate what you think shows that you reduced risk to enter full-scale development.’ Giving the contractor the latitude to fly his test program is a little bit different.*” [JABOUR91] Colonel John M. (Mike) Borky, former director of ATF avionics, noted that the Dem/Val phase was a huge success because it fostered rapid technology insertion and established the baseline configuration for the next phase, FSD. This method of testing along with demonstrations insured that once the selected design went into FSD, there would be no last minute surprises dragging out the schedule. There would also not be the need for additional large sums of money to fix a system that did not work. [BORKY91]

### 7.3.2.5 Performance Incentives

Performance incentives are a proven risk reducing acquisition strategy that rewards developers for exceptional contract performance. For incentive or award type contracts [i.e., cost-plus award fee (CPAF), cost-plus incentive fee (CPIF), and firm-fixed-price incentive fee (FFPIF)], you may have difficulty identifying the factors on which to base the additional fee. While there is no standard guidance, incentive awards can be based on:

- **Milestone completion.** The degree of completeness for major software milestones has been successfully coupled with award fees. Surveys among Air Force managers show these award fees can enhance software development and documentation quality.

**NOTE: Difficulties can arise if software development progress is evaluated independent of system development progress.**

- **Software quality and reliability.** Quality and reliability are prime candidates for award or incentive fees since they greatly affect both development and support. Remember, software quality and reliability can only be determined through an effective measurement program. [HUMPHREY90]

Contractors have a number of corporate goals which include profit, perpetuation, growth, and prestige. Most defense contractors are adverse to risk and operate on the premise that a satisfactory profit at acceptable risk is better than maximum profit at a high risk. Software development for the military is a very high-risk corporate endeavor. Performance incentives and award fee contracts offer a means for motivating contractors to achieve more than minimal program objectives without excessive risk. This forces the Government and the contractor to work as a team — rather than as adversaries.

**CAUTION: A problem with this type approach is that often so much effort is put into preparing for the award fee board that productivity is sacrificed. Also, problems that should be dealt with as a team can be hidden from the Government to look good on performance reports. Nevertheless, award fees are a proven, effective means for assuring the achievement of desired performance, quality, and supportability objectives.**

### 7.3.2.6 Make-or-Buy

The contractor's make-or-buy decision recognizes that few, if any, prime contractors can or want to make all of the many components required for a sophisticated, complex major software-intensive system in the time allowed, within cost limits, and at required quality levels. Buy decisions on the part of the prime can involve buying COTS or employing subcontractors to make subsystem software components. However, as you will learn in Chapter 8, *Contracting for Success*, subcontractors present government managers with a new set of issues. With subcontractors, the program office is divorced from direct contact management of the software developer because the subcontractor is under contract to the prime, not the Government.

The government make-or-buy decision involves whether to buy the software as COTS or contract for the development of a custom (or a combination of custom and COTS) solution. If custom-made, data rights issues must be analyzed and resolved.

### 7.3.2.7 Pre-planned Product Improvement (P3I)

If a technology or threat change occurs during the development of a software-intensive system, you can respond to these changes in one of two ways: (1) redesign the system to incorporate the change, or (2) continue the development as originally designed to deployment and modify the system later in the field. Both of these approaches can be costly to implement. There is always the risk that complete success in meeting unprecedented and unplanned for threats (or needs) will not be achieved. P3I provides an approach to meeting such needs without having to develop a new system. It entails making plans for probable future needs by improving the system as technology becomes available. The advantages of this strategy are:

- Responsiveness to threat changes and future technology development,
- Earlier initial operational capability (IOC) for the baselined system,
- Reduced development risk,
- Potential for subsystem competition,
- Enhanced operational capability for the final system, and
- Increased effective operational life.

The disadvantages are:

- Increased nonrecurring cost during initial development,
- Increased technical requirements in areas such as memory efficiency, source code efficiency, and reliability,
- Increased complexity in configuration management,
- Vulnerability to goldplating accusations and funding cuts,
- Compounding system management problems due to parallel developments, and
- Interference with the orderly development and implementation of effective support plans and procedures.

---

## 7.4 Program Planning Process

The planning process involves decomposition of the system into functional elements or subsystems. The functional subsystems are then decomposed or allocated into lower tier elements. This process continues until the smallest functional element is identified. Within this systems engineering process, various market analyses and trade studies are conducted to determine the best solution to satisfy the particular subsystem allocated requirements or element of the subsystem. Normally, this process follows a hierarchy where the architecture is the first element to be evolved. The architecture is usually composed of components, interface between components and the functions to be interchanged. The architecture design also considers timing and bandwidth of the interfaces. From this architecture, the functional specifications for the components can be developed. These components will typically consist of both hardware and software.

The elements of a Strategic Software Management Plan are:

- **Objectives and scope.** The objectives identify the overall goals of the program, without consideration for how they are accomplished. The scope identifies the primary functions the software must accomplish, defined quantitatively. It describes what has to be done, for whom, by when, as well as the criteria for determining program success.
- **Risk assessment/management.** This activity filters throughout the process by determining in advance the possibility that a problem will occur, estimating its probability, evaluating its impact, and preparing solutions in advance. Risk assessment begins prior to acquisition strategy development and continues as an integral part of software management activities
- **Decomposition of software components by function and task.** Rather than attempting to understand and plan the entire program as a single entity, experience shows that it is easier and more effective to break down the overall program into smaller, more manageable elements. At a top level, the System Segment Specification (SSS) identifies those requirements and system functions that must be fulfilled by either software or hardware. At increasingly lower levels, the work breakdown structure (WBS) subdivides the program into more easily defined, understood, tracked and managed discrete tasks.
- **Market analysis.** This consists of trade studies for hardware and COTS software products. It also entails assessing competitive sources through Sources Sought Announcements, Broad Agency Announcements (BAAs), and Requests for Information (RFIs) in the *Commerce Business Daily*.

- **Resource estimation.** These are quantitative assessments on the number of people required and the cost, schedule, and size of each individual element comprising the whole software development task.
- **Software size estimates.** These estimates are quantitative assessments of the amount of code required for each product element (system, subsystem, component, and/or module).
- **Software cost/schedule estimates.** These estimates are based on the size of the program, product attributes (such as application complexity and security requirements) and environmental considerations (such as development team productivity, CASE tool use, and availability). Various estimating techniques and/or models can be used to estimate manpower requirements (staff-months of effort), cost (\$), and schedule (calendar month duration).
- **Software support estimates.** These are estimates of the resources required after system deployment. They are typically based on system size and original development effort. Although actual post-deployment costs must include product upgrades, error corrections, future evolutionary enhancements, and rehosting to new hardware platforms, most software support estimating models *do not* include all these elements in their estimates.
- **Progress measurement and control.** These are on-going measures after contract award. They consist of formal programs for measuring and evaluating your contractors' progress against baselined budget, schedule, and quality standards. Typically, this involves defining and collecting specific metrics that are consistent with the agreed upon baseline. Although these measures may not answer all the questions about why variations from the baseline are being experienced, they should be sufficient to identify that significant deviations are occurring. They should also provide you with sufficient information with which to question your developer. Historically, *this has been one of the weakest software management activities.*

**ATTENTION! The Strategic Software Management Plan need not be developed by the government program office except in outline form. However, one of the required products for evaluation during source selection should be detailed strategic software management plans submitted by the offerors with their proposals.**

## 7.4.1 Planning Objectives

A veteran of the Vietnam War, Colonel Harry G. Summers, Jr. greatly influenced the serious study of strategic planning within the US Army. His theory on objectives was:

*“The first principle of war is the principle of The Objective. It is the first principle because all else flows from it. It is the strategic equivalent of the mission statement in tactics and we must subject it to the same rigorous analysis as we do the tactical mission.”* [SUMMERS81]

The objective of the software planning process is much like the first principle of war. It is the first activity of the planning effort because all else flows from it. It provides a framework (or plan) from which an understanding of the mission and the execution of the effort can flow. This includes having a baseline upon which to estimate the resources, cost, and schedule required, as well as to evaluate recommendations for program changes. The plan serves as a guide for the Special Program Officer (SPO) and as a means to communicate the content and execution of the program to individuals outside the SPO. As the program progresses, the objective and the plan must be subjected to rigorous analysis. You will progressively be able to quantify your original estimates. Thus, it is important to *update your plan* to reflect any changes in requirements and

program management issues, and to more accurately document your new understanding of costs/schedules, risks, and other technical issues.

---

## 7.4.2 Planning Scope

The first activity in the planning process is to define the scope of the software effort. It includes function, performance, constraints, interfaces, and reliability. The definition process starts with the System/Segment Specification (SSS), which is further decomposed with the work breakdown structure (WBS). Make sure the functions described in the Statement of Scope are evaluated and refined to provide the greatest amount of detail before beginning your estimation process. This means that the performance allocated to the software segment during systems engineering must be bounded and stated explicitly (e.g., quantitative data such as the number of simultaneous users and the maximum allowable response time). Constraints and/or limitations (e.g., cost/weight factors that restrict memory size) are the limits placed on the software by external hardware, available memory, or other existing systems. Mitigating factors (e.g., desired algorithms, well understood and available in Ada) must also be taken into account. [PRESSMAN92]

---

## 7.4.3 Recommendations for Program Planners

Past experience has given us a number of recommendations we can use to produce better, more realistic plans.

- Planners must allow for an extended schedule to compensate for the impacts of implementing new methodologies and adequately train personnel to build a repeatable process.
- Ensure there is enough schedule time to include quality in your product. Allow enough lead time between reviews and formal delivery of documentation.
- When new equipment, methods, and processes are introduced concurrently, a tremendous learning curve exists. Bringing personnel into a program after it starts makes it difficult for them to grasp the scope of the program and become fully productive. Bring on key personnel as early as possible in accordance with the staffing plan. Any other new personnel should be phased in incrementally so as not to interfere with total team productivity and cohesion.
- Scheduling for an integrated government/contractor team must involve both government and contractor management.
- On a program involving new processes, equipment, and methodologies, development personnel must be allowed time to come up to speed on goals, tasks, and associated start and completion dates. This can be accomplished by providing each team member with inchstone schedules that are planned and staffed 60 days in advance. These should include all known leaves, vacations, training, holidays, commander's calls, etc.
- To avoid a series of nontrivial changes during software analysis and design, iron out Interface Requirement Agreements (IRAs) as early as possible. Interfaces can be identified as a technical risk. Organize a Risk Management Working Group to oversee IRAs and a risk mitigation officer to monitor and plan risk mitigation.
- Make sure task configuration management (CM) personnel are briefed on CM policies, procedures, and the possible consequences of not following them. CM policies and procedures should be included in the SDP and distributed as required reading to all task members.

- The configuration requirements, implementation, and maintenance for a local area network (LAN) requires planning and dedicated, specialized personnel. Therefore, upfront analysis and planning to determine LAN requirements is necessary with special consideration paid to how much and what type of terminal equipment will be connected to the LAN. Include communications specialists in your staffing plan.
- Adequate identification of hardware and software constraints, connectivity, and supporting documentation must be planned for prior to starting a task. Proper planning of staffing and tools for each task lessens the impact on productivity.
- Government staffing must be coordinated between the task leader and his/her government counterpart to ensure appropriate skills are provided at proper times in the life cycle. Have each task leader identify a staffing strategy. The technical quality assurance evaluator (TQAE) should then review this strategy and plan a similar strategy in coordination with the task leader.
- An external interface process should be documented in a separate appendix to the SDP.
- Although not usually scheduled for delivery, prototype development requires a good support environment. This includes the hardware environment and regular system backup. These elements, often taken for granted, should be routinely provided in development deliverables.
- Documentation must be available from the onset of any new engineering endeavor. Prototyping language manuals and texts will pay for themselves almost immediately after acquisition, as they eliminate the time-consuming trial and error approach to learning.

---

## 7.5 Program Decomposition

In Chapter 9, *Engineering Software-Intensive Systems*, you will learn that one reason for employing the principles of software engineering is that it provides a method for handling complexity. This is accomplished by applying the old adage of *divide-and-conquer*. Major software developments must be decomposed (broken down into manageable parts) to enable realistic estimates of size, time, and manpower. Methods of decomposition differ depending on program objectives, which may be based on either function or design. Functional decomposition divides the program into basic components from a user's perspective; whereas, design decomposition divides it into software components or modules. [BENNATAN92] Your first layer of decomposition is at the system level with the SSS. From that the WBS is developed.

---

### 7.5.1 System/Segment Specification (SSS)

The most important and critical aspect of system development is to nail down function-level system requirements before they are allocated to either hardware or software components. For weapons systems, the SSS, containing general software requirements, is an initial method of decomposition at the system level. In the SSS, system requirements are defined in quantifiable, measurable, and testable terms. [DSMC90] The SSS is then the basis for further decomposition into the WBS.

## 7.5.2 Work Breakdown Structure (WBS)

A WBS should be developed for each major acquisition program (or major modifications thereof), and for each individual contract within the program. The WBS, in its various forms, can serve as a useful tool for planning, control, and communication throughout your program. The WBS, if properly written, defines the program's total objectives and relates the many work efforts to the overall system. The WBS is the foundation for:

- Program and technical planning,
- Technical description of program pieces,
- Cost estimation and budget formulation,
- Schedule definition,
- Statements of Work and specification of contract line items,
- Progress status reporting and problem analysis,
- Tracking of technical changes [Engineering Change Proposals (ECPs)], and
- Engineering management.

### 7.5.2.1 WBS Interrelationships

The summary WBS defines the upper three levels of a system. The project summary WBS is tailored to a specific program or project. The contract WBS defines the complete work effort for a particular contract or other procurement action. It contains applicable portions of the project summary WBS plus the extension of any levels necessary for planning and control. The interrelationship between WBSs is illustrated in Figure 7-2.

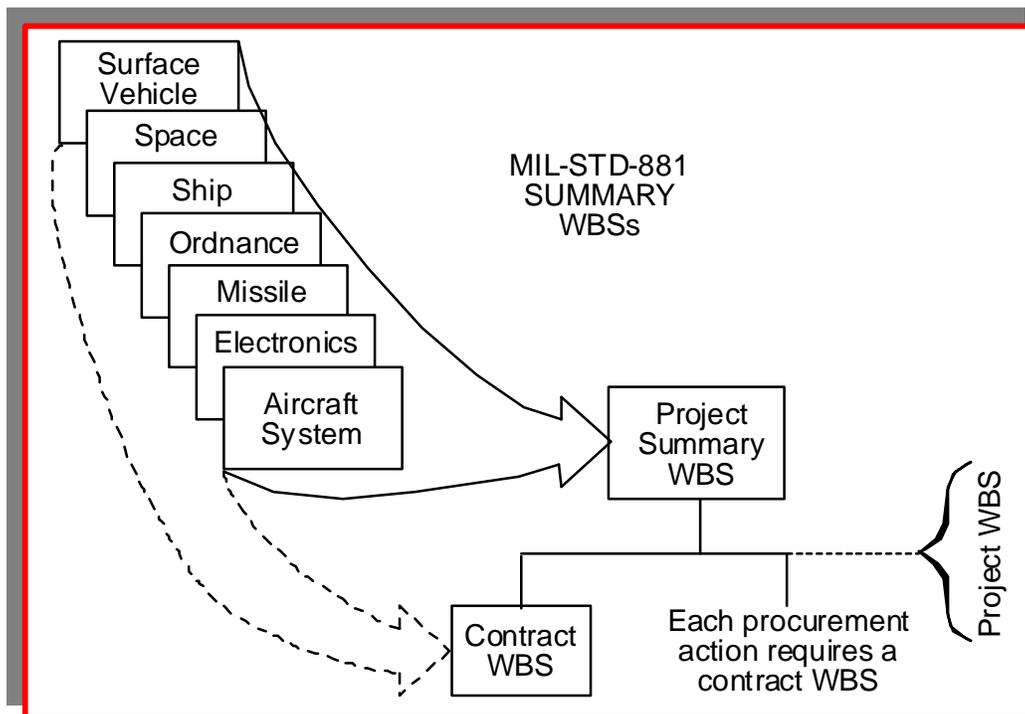


Figure 7-2. Interrelationships Among WBS Types

From the project summary WBS, individual contract (or development organization) WBSs can then be developed. The program office can initiate preliminary contract WBSs before contract award that contain contract line items, configuration items, contract specifications, and industry responses to the draft RFP. The initial project summary WBS and first contract WBS must be established at the award of the first development contract. As the program progresses and additional contracts are let, the project WBS must be extended to all levels it addresses, but the basic structure should remain unchanged. A single project WBS, with element nomenclature and definitions, should be maintained throughout the acquisition process to ensure traceability. The components of the WBS are:

- **Prime mission product.** The prime mission product element is the hardware and software used to accomplish the primary mission of a defense materiel item. It includes all integration, assembly, test, and checkout, as well as, all the technical and management activities associated with individual hardware and software items.
- **WBS element.** This describes a discrete portion of a WBS that is either an identifiable item of hardware, a set of data, or a service. An element can consist of one or many work packages.
- **Subsystem.** This refers to all the hardware and software components of a subsystem.
- **Software component.** This is all the software integral to any specific subsystem specification and can be an aggregate of application and system software [*discussed below*]. (It excludes software specifically designed and developed for system test and evaluation.)
- **Work package.** This represents the work to be performed at the lowest WBS level where work performance is managed. Developed by the contractor, it defines the work, how its accomplishment is measured, how it is tied to a schedule, and where responsibility lies for production of the operating unit. Interrelating the who, what, when, and how much for any task effort, it is the heart of management control and provides visibility at designated levels. Program performance can be measured and controlled by monitoring reports on the technical and schedule aspects of each work package or combination of work packages.

### 7.5.2.2 Prime Mission Product Summary WBS

The prime mission product summary WBS identifies the upper three levels of a WBS and defines the top-level software elements and their placement in the structure. The prime mission product summary WBS is used to develop the software project summary WBS.

### 7.5.2.3 Software Project Summary WBS

A software project summary WBS is usually the result of the systems engineering efforts conducted during Concept Exploration. At this time, the most suitable summary WBS software items are considered that best satisfy operational needs. The preliminary software summary WBS should not be constraining, but evolves and is tailored as program objectives stabilize. Developers should be encouraged to propose changes to the preliminary project summary through creative, alternative development options. The software project summary WBS elements are:

- **Software WBS elements.** Software WBS elements are described generically and apply to each type of defense system. The associated activities and deliverables for which cost data are collected are listed with each software WBS description.
- **Application software.** Application software is specifically developed for the functional use of a computer system. Examples are battle management, weapons control, and data base management software. This element refers to all the effort required to design, develop, integrate, and checkout prime mission product applications, builds, and CSCIs. It excludes all software integral to any specific hardware subsystem specification. Figure 7-3 illustrates the breakdown of both application and system software CSCIs.
- **System software.** System (or support) software is designed for a specific software system, or family of software systems, to facilitate its (and its associated applications; i.e., operating systems, compilers, and utilities) development, operation, and maintenance. It also includes all the effort required to design, develop, integrate, and checkout the system software, including all software developed to support any prime mission product software development. It can also include multiple builds.

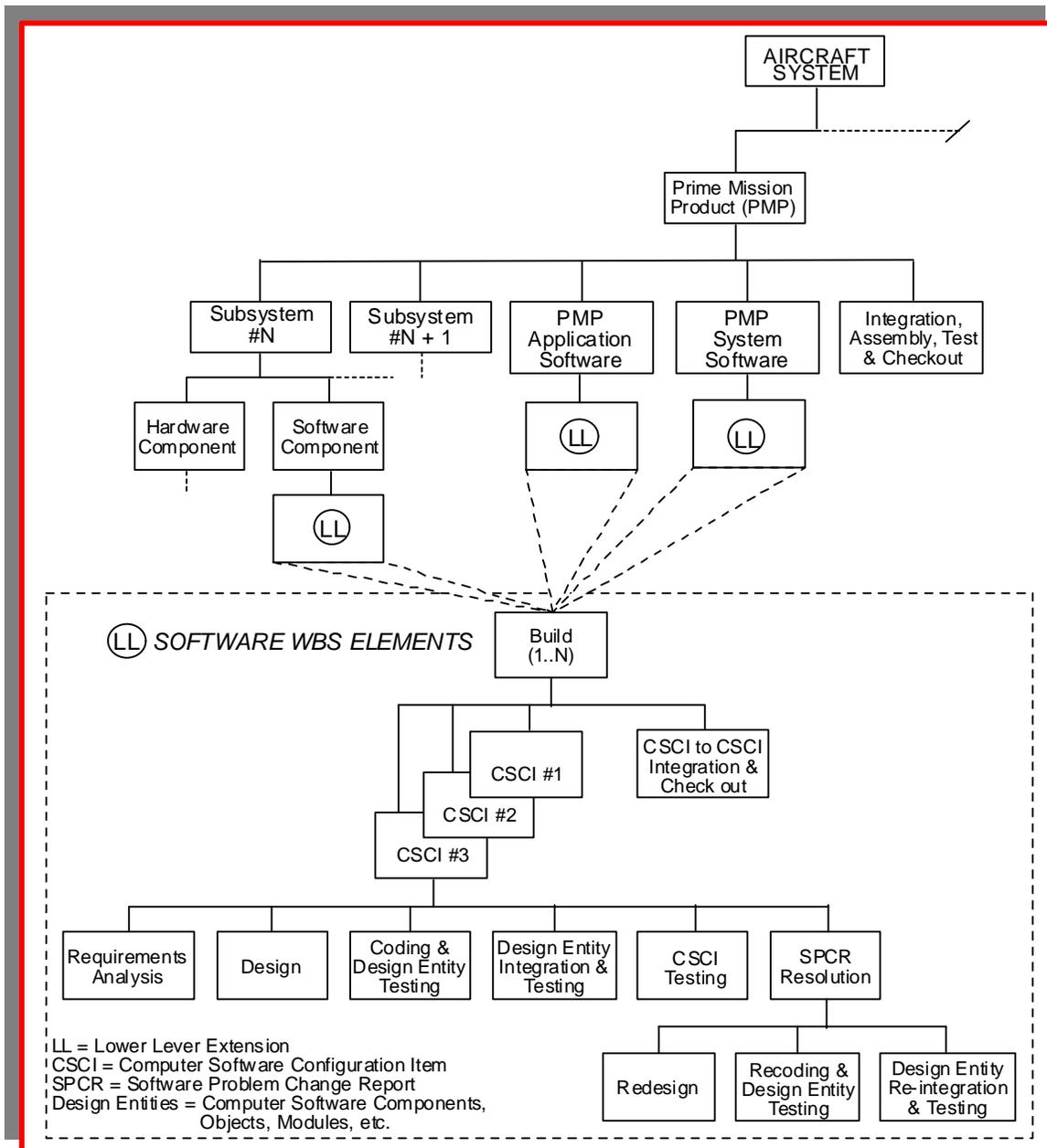


Figure 7-3. Interrelationships Among WBS Types

### 7.5.2.4 Software Contract WBS

Only one preliminary software contract WBS is used for each RFP and its ensuing contract WBS. The program office structures a preliminary contract WBS by selecting elements of the approved project summary WBS that apply to that contract. It then organizes them into a framework supporting the approved project summary WBS and development objectives. Software subsystems may then be extended to the next lower level. Traceable summarization of individual contract WBS(s) into the approved project summary WBS are maintained. The contract WBS does not need to completely mirror the project WBS. For contracting issues (e.g., cost accounting) a WBS different from the project tracking WBS may be necessary. The functional integration of the project summary WBS with the contract WBS is illustrated in Figure 7-4.

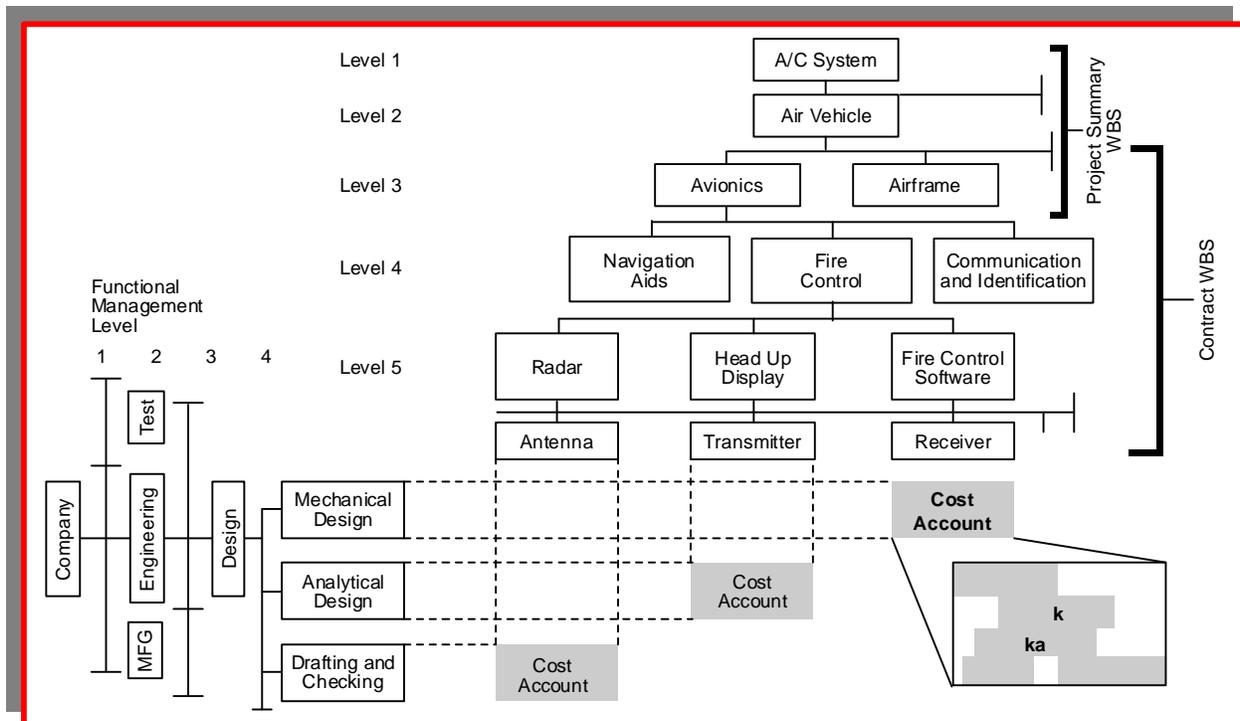


Figure 7-4. Software Project and Contract WBS Functional Integration

In their proposals, or during source selection, offerors are encouraged suggest changes to certain elements to make the contract WBS more effective. These changes are approved by the government program manager. The final contract WBS, based on the contractor's proposal, suggested changes, and contract negotiations, becomes the basis for a more detailed definition necessary to manage the effort. The contractor must also prepare program-specific terminology and definitions for extended elements of the contract WBS.

A couple of points must be emphasized. First, the contract WBS provides the link between the contracted effort and the overall program to include description of the interfaces necessary to integrate the software of one contractor with that of other contractors or agencies. This is to ensure that all software being developed is compatible when integrated with other software and hardware at the next higher level of integration. Second, be careful to select WBS elements that permit structuring of budgets and tracking of costs to whatever level is necessary for control.

You can accomplish this by assigning job orders (or customer orders) to the cost account level for in-house efforts and by structuring line items (contract data requirements list (CDRLs)) or work assignments [discussed in Chapter 8, *Contracting for Success*] in accordance with the WBS. Usually, a cost account is established at the lowest level of the contract WBS where costs are recorded and compared with budgeted costs. This cost account (WBS element) is a natural control point for cost/schedule planning as it is the responsibility of a single organizational element. Contractors should maintain records to the work package level so the Government has visibility to the cost account level. *Ideally, you and the contractor will agree on a WBS which is integral to [and not disruptive of] their development process that they would normally use for internal tracking and management.*

### 7.5.2.5 Software Project WBS

The software program/project office prepares the software project WBS by compiling the elements of the extended contract WBS(s) with the project summary WBS. The program office then incorporates the levels of the extended contract WBS(s) it considers necessary for program management and other related requirements into the project WBS. This compilation occurs as successive extensions of the individual contract WBS(s) are identified throughout the program. The formal project WBS is completed prior to initiation of the system integration and test phase. A 3-level project summary WBS for the F-22 is illustrated in Figure 7-5.

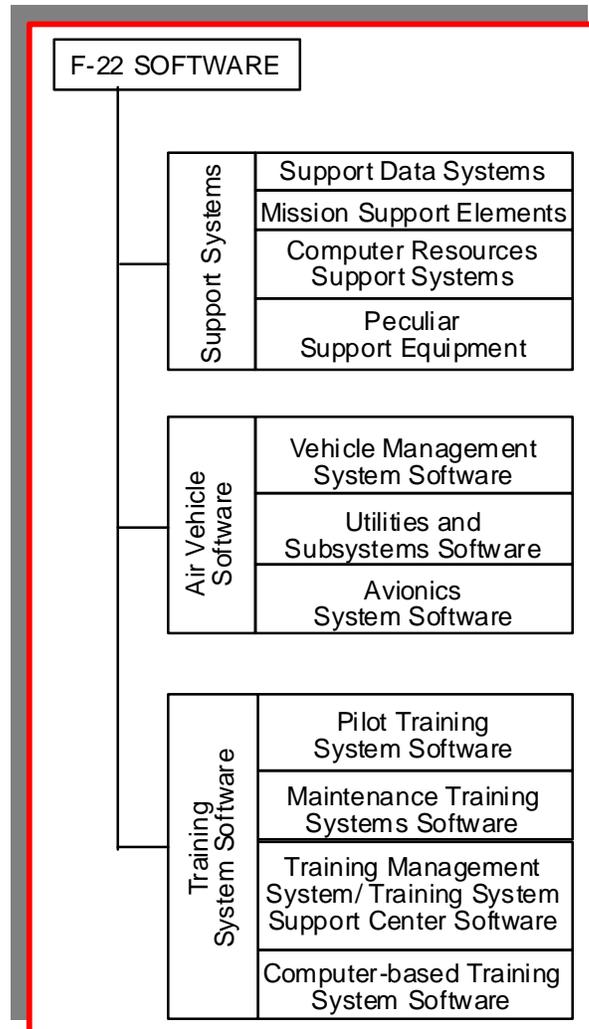


Figure 7-5. F-22 3-level WBS

The Integrated Master Plan/Integrated Master Schedule concept that evolved on the ATF program is one approach to force contractors to perform detailed, step-by-step planning and report progress and costs against their plan. If contractors are held accountable to deliver functions rather than configuration items, they cannot cheat on software performance and integration because delivering the hardware alone will not trigger payment. This is something of an oversimplification, but the basic concept of requiring that contractors deliver integrated system capabilities and minimizing progress payments for arrival on dock of a system capability, rather than bits and pieces, is highly recommended.

**WARNING TO WEAPON SYSTEM PROGRAMS! The lack of a software WBS has been the Achilles Heel of many weapon programs. Do not be caught without meaningful insight into your highest risk area.**

## 7.6 Market Analysis

The Defense Standards Improvement Council (DSIC) states that market analysis is key to meeting the spirit and letter of Secretary Perry's June 1994 Memo. [See Chapter 4, *DoD Software Acquisition Environment*.] You should perform a market analysis to determine if commercial products are available that meet your identified need because current MilSpec reforms make aggressive market analysis imperative.

Through comprehensive market analysis, you will be able to ascertain if adequate commercial product alternatives exist and to identify satisfactory replacements for software MilSpecs and MilStds. If your market analysis illustrates that certain software MilSpecs or -Stds can meet your identified need, your analysis results can serve as the basis for a waiver request to the Milestone Decision Authority (MDA), or you can cite the MilSpecs or MilStds as guides without mandating that they be literally followed.

Whether your program is a new-start, an on-going, or one in post-deployment software support (PDSS), you must perform a market analysis prior to every requirements definition effort. The data you collect during the market survey are then used to reassess your original requirement. You must determine whether a modification to the original requirement will result in greater overall value to the Government in terms of cost, performance, availability, reliability — or other risk drivers you have identified. Your market survey should also cover maintenance and support data, test results, and user satisfaction analyses. These data are used in developing your support strategy and the TEMP. [The SD-5, "Market Analysis for Nondevelopmental Items," Assistant Secretary of Defense (Economic Security) [OASD/(ES)] describes a generic approach for market analysis. A training program is also available from OASD/(ES). See Volume 2, Appendix A for a point of contact.]

### 7.6.1 Software Product Definition and Decomposition

Software product definition and decomposition will be complete once you have accomplished the basic planning process discussed above. Your product will be identified and decomposed, at least initially, through the SSS and the various WBSs. However, it may be necessary to modify or adapt these items to your software cost, schedule, resources, and support estimate preparation requirements.

## 7.7 Baseline Estimates

The basic software estimating process mirrors the strategic planning process and builds on and supports many of the other planning steps. It consists in defining what will be estimated, breaking the total effort into appropriate lower-level elements, determining the scope (size) of each element, assessing the software development environment, and performing assessments of alternatives and risk factors. Once these elements have been quantified, evaluated, and boundaries placed around their values, baseline estimates of cost, schedule, resources, and support can be determined and assessed. Table 7-1 (from Kile's *A Process View of Software Estimation*) outlines the steps necessary for bid preparation. Although it names the steps and presents the view differently than discussed here, the basic process is the same. [KILE91]

Phase	Major Activity	Specific Products
1. Design Baseline	Define a point of sufficient precision to identify the number of CSCIs and the required functionality of each.	List of CSCIs, functionality, and similar completed projects or CSCIs.
2. Size Baseline	Using the products from the Design phase, define the expected size for each CSCI.	List of CSCIs with appropriate size information.
3. Environmental Baseline	Using the products from the two previous phases, determine the environmental characteristics required and available to perform the job.	List of software cost model parameters and their initial settings along with a written rationale for each.
4. Software Baseline Estimate	Using the size and environment products, make a software cost model run (using whatever model best satisfies the organization's needs).	Output from the software cost model showing schedule and cost information.
5. Project Baseline	Using the output from the Software Baseline Estimate phase, add those elements not included in the particular software cost model (each model has a specific set of items not included in the estimate) and subtract those elements excluded from this project.	A complete estimate of the costs and schedule for the software portion of the project.
6. Risk Analysis	Determine the cost/schedule risk associated with the Project Estimate. Make changes to the size or environment products to perform what-if analyses. Determine the size and/or environment setting required to validate the final software bid.	Risk assessment, risk graphs, risk memorandum with Parameter-by-Parameter risk explanations.
7. Project Bid	Perform analysis of the Project Estimate, considering such factors as expected competition, type of contract, budgetary or personnel constraints, risk analysis, etc. Convert labor and other direct charge (ODC) estimates into contractor's price and determine the Project Bid.	Project Bid.
8. Dynamic Cost Projection	Using existing known environment and size information, produce a revised Project Estimate and determine the remaining costs and schedule-to-complete for the on-going project.	Cost-to-Complete, Schedule-to-Complete, Size-to-Cost.

Table 7-1. The Software Estimation Process

The software estimating process is an interactive, dynamic process. As program requirements, the development environment, and/or the program funding profile change, *re-estimation of the effort/cost and schedule must be performed*. Contractor Engineering Change Proposals (ECPs) must also be evaluated for their affect on both development and support costs, as well as schedule. Funding constraints typically result in program delays which can, in turn, increase cost.

To develop master schedules, acquisition strategies, and preliminary budgets, a preliminary cost/schedule estimate reflecting the program baseline needs to be developed using preliminary size and environmental assessments. This baseline estimate provides a starting point from which alternatives may be compared and changes tracked. Throughout development, as assessments are updated to reflect current conditions, cost/schedule estimates must be updated to support decision making at all levels. A cost track from the baseline estimate to each update, as well as clear, understandable documentation substantiates the need for programmatic change. *[A rule of thumb for a well-documented estimate is that it is verified by a second party.]* The goodness of an estimate depends on whether factor assessments are realistic, appropriate risk is considered, and estimating methodologies substantiate reasonableness of the cost estimate. Significant cost and schedule *drivers* should be re-estimated and documented using a secondary methodology as a confidence check. A minimal confidence (or sanity) check, is performed on significant cost elements to assure that the estimation is within an acceptable range of general knowledge (e.g., sources lines-of-code (SLOC)/staff month is within range of similar software programs).

**NOTE: An overview of several estimating techniques/methodologies is found in Chapter 13, Software Estimation, Measurement, and Metrics.**

The preliminary estimate becomes the baseline from which the process of updating your estimates proceeds, and continues throughout the development life cycle. As program knowledge increases, metrics data are collected and analyzed, and your estimates are updated, your cost and schedule estimates will become increasingly more accurate. This approach does not omit or conflict with longer term acquisition strategies such as systematic reuse and families of product-lines and systems. Long term, wide scope acquisition planning is necessary to ensure a cost-effective acquisition, especially when such requirements may contradict the profit motives of individual development contractors.

Once an initial cost and schedule estimate has been developed, significant effort is required to analyze and understand the estimate before accepting it as a formal part of your strategic plan. The analysis of your original estimate serves four purposes: (1) to make sure the estimate is thoroughly understood, (2) to insure that the estimate is as accurate as possible; (3) to provide a baseline upon which to evaluate programmatic alternatives (e.g., trade studies, software capability assessments, tradeoff analyses, and development methodologies), and (4) to conduct risk analyses. The original estimate analysis includes answering the following:

- Does the estimate make sense?
- Are estimated schedules, costs, and effort consistent with prior experience?
- Does the estimated effort, cost, and schedule meet programmatic requirements?
- Are required productivity levels reasonable?
- Have all relevant costs been included?
- Have any cost elements accidentally been included more than once because different estimating techniques were used for different WBS elements?

---

## 7.7.1 Estimation Accuracy

The accuracy of your cost estimate directly relates to the quality of the information upon which it is based. The exactness of this data increases as a function of time and the stability of requirements. During the planning phase of a program, requirement uncertainties often result in questionable estimates. As time progresses, the fidelity of the information improves along with the accuracy of the estimates. [MARCINIAK90] The quality of the information is also dependent on the skill and experience of your analysts/engineers who gather and analyze the input information. The quality of the estimate is, similarly, dependent on their skills and experience in software cost and schedule estimating, the specific estimating methods and models used, as well as their familiarity with the software system being estimated.

Using a second (or possibly third) estimating technique or model to identify these potential problems is a proven, effective way to compare estimation results (after normalizing results for equivalent content). To correct any identified problems, your cost analyst must change model input settings [*not input data*] to reflect a better understanding of the information required by the model and/or seek additional clarifying data upon which to base changes in model inputs. If your model's estimated schedule exceeds programmatic requirements, your analyst may need to *turn-on* a schedule constraint variable within the model. If the staffing profile predicted by the model is inconsistent with the development plan, a staffing constraint variable may need to be adjusted. After several iterations (under a variety of assumptions and with varying parameter settings) your analyst should arrive at an estimate that is both credible and reasonably accurate. This estimate should also include a risk assessment. At this point, the estimate may consist of a range of estimates that reflect different assumptions and probabilities of success rather than a single-point-estimate. You should review these estimates and risk assessments and provide additional guidance for further analysis and/or approval.

Once the estimate is as accurate as possible using known information, it can be used to perform sensitivity analyses, risk analyses, and *what-if* exercises by varying model inputs based on expectations or alternative sets of assumptions. If there is uncertainty about the size measurement (or other factors influencing the cost or schedule), high and low end estimates of the expected range should be developed. These studies can be used to identify risk areas and to develop contingency plans. If there are constraints on your budget or schedule, your estimate should be derived taking these limitations into account. This will provide your baseline estimate as to the viability of developing the software within identified constraints. If it appears that either cost, schedule, or resource limitations can not be met, other programmatic options must be examined during planning, rather than waiting until these constraints have been violated. *It is extremely important that you do not change estimate parameters to meet programmatic restrictions.* This will merely invalidate your estimate, your program planning activities, and greatly reduce your likelihood for program success.

**CAUTION! Beware of “*Rosy Scenario*” (also known as “*Optimism*”). Program managers are inclined to manipulate the input variables to software estimating models to assure an “*acceptable*” outcome in terms of estimated cost and schedule. One example is to understate the size of the program, while another typical situation is to overstate the capabilities and/or resources (tools and practices) of the development team. Time and again this has led to broken programs, delays, restarts, loss of confidence, all around embarrassment, and on occasion — program cancellations. Well-documented estimates using reasonable, but conservative, assumptions will bring you accolades in the long run, even if there are grimaces and groans in the near term about the predicted cost and schedules taking too long.**

**NOTE: All analysis and estimation to this point have been done by the acquirer. This analysis may have NO relationship to a similar analysis performed by the developer. The analysis should provide the acquirer with a rough order of magnitude estimate of what the development will take in terms of cost and schedule. However, it is the developer’s estimate of cost and schedule that ultimately count!**

### 7.7.1.1 Program Estimate Selection

After the analysis of the estimate is complete, it is up to you to select the cost and schedule estimate baselines which become part of your Strategic Plan. Ideally, this is performed in cooperation with your development team manager as an on-going activity in your program management process. This baseline must be periodically updated to reflect changes in the Strategic Plan as more is known about your program.

The key element in selecting the baseline estimate is the level of cost and schedule risk you are willing to accept. You need to understand that, although it may be possible to accomplish your program within the cost and schedule to which you have committed, based on historical examples there may be only a 10% probability you will succeed. It is only through realistic estimates and early planning that this probability can be increased. Similarly, if the schedule for a software development is dictated by other mission-critical factors (such as a payload launch date), you must realize and understand the probability of meeting that schedule. Once you understand your probability of success, you can rethink your strategy by planning various incremental efforts to insure that critically-necessary functionality is completed on time. Other noncritical functionality may be deferred to later development stages, or if necessary — omitted completely. Software estimating models (used in concert with independent risk analysis techniques) should be used in assessing the critical cost and schedule risks associated with changes in your program.

---

## 7.8 Continuous Program Planning

The final and most challenging step in planning is for you to constantly re-implement the planning process throughout the life of your program. Budget cuts, personnel cuts, short schedules, incessantly changing requirements, and the development environment force you to continuously re-evaluate your estimates. To ensure successful program completion, you need to update your Acquisition Plan. Figure 7-6 illustrates how planning is a continuous, iterative process.

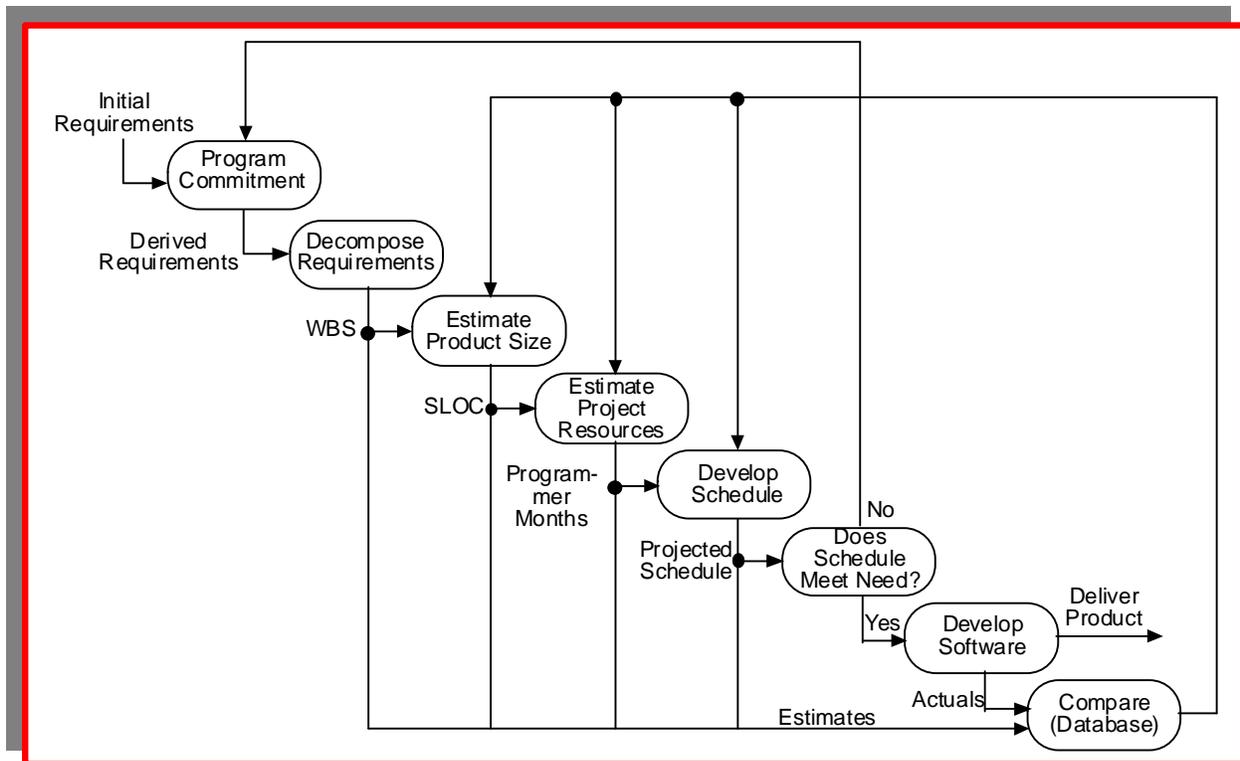


Figure 7-6. Iterative Software Planning Process [HUMPHREY89]

Remember, *the requirement to prepare software cost and schedule projections does not end.* The baseline estimate must be updated to reflect changes in your program environment, your increased program understanding, and the actual metric data being collected. Similarly, cost and schedule impacts of proposed and unforeseen changes can be quantitatively evaluated using estimating models and your baseline estimate. These may not answer all the questions you have about what is happening in your program, but they will provide you with a solid starting point from which you can direct your questions.

Software estimation (and indeed software development) is still an art practiced with varying sets of standard procedures, tools, and methods. Also, there are many unknown and dynamic variables (i.e., human, technical, and political) in the development process that affect the software effort. For example, requirements are frequently added or changed well into the software testing phase. Because requirements frequently change, there is an axiom in the cost analysis community that states: *“original estimates are never correct because we never build what was originally estimated.”* Also, initial estimates of software size are typically based on limited information and are often driven by *optimistic, rosy scenario*, success-oriented influences. Software cost and schedule estimates also fall-short because the analyst performing the estimate is unfamiliar with either the estimating model(s) being used or the specific details of the program — or both. As a result, estimating the cost, effort, and schedule of software development is a necessary but inexact science. Software program estimates can be improved, however, by using a systematic, disciplined estimation procedures.

---

## 7.8.1 Continuous Planning Recommendations

- No one likes to be a slave to schedule, but adherence to schedule promotes program stability and inhibits requirements creep. Develop your deliverables around the delivery of a functional capability and use cost payments as incentives to meet those deliveries.
- If a software development effort begins to sink because of schedule slips, throwing more people onboard may not help. In fact, more people can actually cause the program to fall even farther behind due to added communications and training requirements that decrease productivity. Therefore, plan team composition and buildup in the early strategic planning phases.
- Initially, QUALITY does not stand out like this. If you do not think quality upfront, you will pay dearly for its neglect later.
- If you are living with evolutionary requirements, it makes sense to pursue an evolutionary acquisition strategy. Develop an acquisition strategy that is flexible, can accommodate evolutionary change, and deal with risk.
- Software planning is an iterative and continuous process. Initial estimates must be refreshed and reflected in updated schedules and resource commitments.
- Monetary reward is a proven incentive for contractors to produce quality software. Money might not be everything, but it sure is way ahead of whatever is in second place.
- Software size estimates have traditionally been poor and rank right up there with estimates on the Gross National Product. Inaccurate estimates of SLOC have been a major impediment to accurate software development cost estimates. Another major impediment has been the failure to accurately estimate the capability of the development team.
- To make your estimates more accurate, use a combination of estimation techniques. Of all the program risks with which you must deal, size estimating will be your biggest planning problem. A word of caution is to use well-documented estimates based on conservative assumptions.
- Use the Mitre Skills Matrix to reduce the team capabilities risk element.
- Identify software support requirements, as well as computer hardware support requirements, in program budgets and schedules.
- A well-planned measurement program is an investment in successful management and product quality.
- Performance risk can be reduced by planning for delivery of incremental levels of functionality.
- Use program decomposition and program management automated tools to get a handle on program complexity.
- When comparing estimates produced by different cost models, make sure they have the same definitions of environmental input parameters.

---

## 7.9 Other Planning Considerations

There are several other areas of strategic planning that you must include in your planning process. These include:

- Use of milestones and baselines to track program progress towards achieving objectives,
- Factoring in the often hidden (but often substantial) cost of software scrap and rework in your estimates,
- Program budgeting and funding considerations,
- Upfront definition of requirements for software safety and security, and
- Planning for future changes in technology that can impact your development efforts.

---

### 7.9.1 Major Milestones and Baselines

Milestones signify major events in the software development process. The completion of requirements and design specifications are major milestone events. Major program milestones often gain added importance through their linkage to other events, such as budget payments used as measures of program progress or for determining baselines. If milestones can be described as major program events, then baselines can be described as major milestones. [BENNATAN92]

The IEEE definition of a baseline is “*a formally agreed upon specification that serves as the basis for further development.*” [IEEE87] Baselines are important in DoD software development as they indicate critical times when major milestones are finalized. Baselines also provide significant and complementary ways to control acquisition programs. Strategic planning baselines include:

- **Cost/schedule control performance measurement baseline.** This baseline provides the budgeted cost of work scheduled and is the measure against which schedule and cost variances are calculated.
- **Configuration management baselines.** The software configuration management process is important in providing support to the baselining of system products, and is central for controlling the development process. Baselines that mark the completion of major milestone activities are formal baselines. Changes to formal system baselines can directly impact both cost and schedule. With formal control, any changes to the baselined system must be approved by the authority responsible for system integrity as defined in that baseline. In software development, there are three formal baselines.
- **Functional baseline.** The functional baseline establishes the requirements the system must satisfy. With functional baseline establishment, system specifications are placed under control.
- **Allocated baseline.** The allocated baseline marks the end of the software analysis phase. The allocated baseline is established when requirements are allocated to individual software subsystems. It captures the linkage between the architecture and software requirements.
- **Product baseline.** A product baseline is established when the software system is fully designed, developed, and tested. This baseline defines the produced software product, and provides the framework for modifying the system through defect correction and incorporation of new requirements.

- **Acquisition program baseline (APB).** The APB provides quantifiable targets for key performance, cost, and schedule parameters of an acquisition program throughout the acquisition process phases. The APB has two components for each parameter, an objective and a threshold. Objectives and thresholds are determined differently for cost, schedule, and performance. The user's Operational Readiness Document (ORD) provides performance objectives and thresholds. The ORD also provides the user's requirement for initial operational capability (IOC) and full operational capability (FOC) — both of which have schedule implications. Cost and schedule objectives and thresholds are developed by your acquisition team. [Recommendations for ORD preparation are found in Volume 2, Appendix T.] APBs are sequentially refined as we move through the life cycle phases and are submitted at Milestones I, II, and III. The APB may be adjusted at milestone approval (or program reviews) based on changes in requirements and/or on the results of activities taking place in the previous phase. The APB can also be adjusted in response to a baseline breach. Only those performance, schedule, and cost parameters attributable to the breach, however, can be adjusted.
- **Operational performance thresholds** are the user's minimum acceptable requirement for the system when fielded and are derived directly from the ORD. (Other performance thresholds may be added by the Milestone Decision Authority.) Cost objectives and thresholds should reflect the independent cost estimate (ICE) for the program to meet performance objectives. For schedule, the objective is the most likely date for a key event (such as a milestone review, design review, or the completion of a test activity).

When the operating command identifies an unfulfilled need, they must start working closely with the developing command and the supporting command in defining system thresholds (*minimally acceptable requirements*) and objectives. This approach recognizes that technology, funding, or schedule may preclude the developing command (and its contractors) from achieving each and every objective. The objectives, if properly integrated into the program, can help the system designer accommodate P3I. Thereby, parallel development upgrades can be incorporated at appropriate procurement stages. A better understanding of requirements and more effective teamwork can be achieved while still maintaining acquisition competition.

Program executive officers and senior acquisition executives must establish an atmosphere that fosters frank program assessments by the development team. One method is to *actively support realistic realignment of system requirements and schedules during program reviews*. This must be a strategic part of the iterative process of refining the system by considering technology, budget, and schedule. Program baseline documents (e.g., requirements correlation matrices and system maturity matrices) must mature throughout the evolutionary process and be used as management tools. They should reflect the refinement of system requirements as the development proceeds, provide an audit trail of requirements, and establish the rationale behind subsequent requirements changes.

During each phase of development, you need to maintain a current estimate of cost, schedule, and performance parameters. If the current estimate indicates a threshold breach is anticipated, or has occurred, they must be reported and acted upon.

---

## 7.9.2 Program Budgeting and Funding

The Air Force develops its programs through the Planning, Programming, and Budgeting System (PPBS). It consists of three parts:

- **Planning** identifies the threat facing the nation for the next 5-15 years, assesses our capability to counter it, and recommends forces necessary to defeat it.
- **Programming** allocates resources for competing requirements within the fiscal and manpower ceilings imposed by the Congress. This effort develops a five-year program, i.e., the Program Objective Memorandum (POM).
- **Budgeting** provides the initial estimated cost of approved plans and programs and refines estimated costs as programs are better defined or modified in subsequent POM cycles, budget estimate submissions (BESs), or the President's Budget (PB). [Refer to [AFI 65-601, Vol 3](#), The Air Force Budget Corporate Process]

As a program manager, your role in the PPBS process is important. You may not be involved in the initial planning process, but you are an important player in the software-intensive systems programmed as a result of this planning. You must investigate the technology and software solution recommended to satisfy the planning requirement, and if the solution is not sound, you must bring that to the attention of all concerned and work to resolve all issues. These efforts are critical to the success of your program. In the budgeting phase, your program's costs are tied to the rest of the Air Force's monetary needs for coming years and your program is prioritized relative to its importance and the current probability that it will be successfully fielded. [Refer to *HQ USAF/PE Primer, The Planning, Programming, and Budgeting System, HQ USAF/PE.*] In the past several years GAO has noted that virtually *not one program has received full funding*. As a result, program managers have been forced into the position of having to restructure their programs "on-the-fly." Your challenge in today's environment is to structure your software development program to respond readily and aggressively to uncertain funding. [GAO91]

Field Marshall Erwin Rommel defined success on the battlefield as the ability to be flexible and adapt to volatile wartime conditions. He explained that,

*"Success comes most readily to the commander whose ideas have not been canalized into any one fixed channel, but can develop freely from the conditions around him."* [ROMMEL53]

Success in planning for software management is the ability to stick to your plan while having a plan flexible enough to adapt to changes in the development environment. The planning continues throughout the software life cycle and is one of the most crucial activities you must perform as a manager.

---

## 7.10 References

- [BENNATAN92] Bennatan, E.M., *On Time, Within Budget: Software Project Management Practices and Techniques*, QED Publishing Group, Boston, 1992
- [BORKY91] Borky, Col John M., communication to SAF/AQK regarding draft AFPAM 63-116, December 11, 1991
- [DEMING86] Deming, W. Edwards, Foreword to M. Walton, *The Deming Management Method*, Dodd, Mead & Co., New York, 1986
- [DEUTCH93] Deutch, John, as quoted by John Moore, "CIM Will Play Key Role in NPR Challenge, Paige Declares," *Federal Computer Week*, September 20, 1993
- [DSMC89] *Using Commercial Practices in DoD Acquisition: A Page from Industry's Playbook*, report of the Defense Systems Management College 1988-89 Military Research Fellows, Fort Belvoir, Virginia, 1989
- [DSMC90] Caro, Lt Col Israel I., et al., *Mission Critical Computer Resources Management Guide*, Defense Systems Management College, Fort Belvoir, Virginia, 1990
- [EASTERBROOK92] Easterbrook, Gregg, "Stealth-Creators," *The New Republic*, January 6 & 13, 1992
- [GAO86] General Accounting Office, *Sergeant York: Concerns about the Army's Accelerated Acquisition Strategy*, Report to the Chairman, Committee on Governmental Affairs, United States Senate, GAO/NSIAD-86-89, May 1986
- [GAO91] "Memorandum: Comments on Successful Acquisition of Computer Dominated Systems and Major Software Developments," US Government Accounting Office, Washington, D.C., January 25, 1991
- [HUGHES92] Hughes, David, "Digital Automates F-22 Software Development with Comprehensive Computerized Network," *Aviation Week & Space Technology*, February 10, 1992
- [HUMPHREY90] Humphrey, Watts S., *Managing the Software Process*, The SEI Series in Software Engineering, Addison-Wesley Publishing Company, Inc., 1989
- [IEEE87] IEEE Standard 1058.101987, *Standard for Software Project Management Plans*, Institute of Electrical and Electronics Engineers, Inc., New York, 1987
- [JABOUR91] Jaybour, Lt Col W. Jay as quoted by Michael A. Dornheim, "Air Force's Hands-Off Approach Speeded ATF Testing Programs," *Aviation Week & Space Technology*, July 1, 1991
- [KEMP94] Kemp, Dan, "CSC Software Development at Syntex: A Case Study," briefing, 1994
- [KILE91] Kile, Maj Raymond L, USAFR, *A Process View of Software Estimation*, HQ United States Air Force/C<sup>4</sup> Plans and Policy, Washington, D.C., June 1991
- [KINDL92] Kindl, LTC Mark R., *Software Quality and Testing: What DoD Can Learn from Commercial Practices*, US Army Institute for Research in Management Information, Communications, and Computer Sciences, Georgia Institute of Technology, Atlanta, Georgia, August 31, 1992
- [MARCINIAK90] Marciniak, John J. and Donald J. Reifer, *Software Acquisition Management: Managing the Acquisition of Custom Software Systems*, John Wiley & Sons, Inc., New York, 1990
- [MRAZ91] Mraz, Stephen J., "Face-off Over Tomorrow's Fighter," *Machine Design*, March 7, 1991
- [PRESSMAN92] Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, Third Edition, McGraw-Hill, Inc., New York, 1992
- [REILY92] Reily, Lucy, "Arms Software Hits Flak: GAO Targets Pentagon on Costs and Scheduling," *Washington Technology*, August 27, 1992
- [RICE91] Rice, Secretary Donald B., as quoted by Patricia A. Gilmartin, "US Lawmakers Tighten Scrutiny of B-1B and C-17 Aircraft Programs," *Aviation Week & Space Technology*, March 4, 1991
- [ROMMEL53] Rommel, Field Marshall Erwin, B.H. Liddel Hart, ed., *The Rommel Papers*, Harcourt Brace & Company, New York, 1953

- [ROONEY90] Rooney, Thomas R., as quoted in "ATF Avionics Met Dem/Val Goals, Providing Data for Flight Tests," *Aviation Week & Space Technology*, September 24, 1990
- [SCHWARZKOPF88] Schwarzkopf, GEN H. Norman, "Food for Thought," *How They Fight*, 1988
- [SUMMERS81] Summers, COL Harry G., *On Strategy: The Vietnam War in Context*, US Army War College, Carlisle Barracks, Pennsylvania, 1981
- [SUN500BC] Sun Tzu, Samuel Grifford, ed., *The Art of War*, Oxford University Press, New York, 1969
- [VESSEY84] Vessey, GEN John W., as quoted in the *New York Times*, July 15, 1984
- [WORDEN94] Worden, Col Simon P. and Lt Col Jess M. Spanoable, "Management on the Fast Track," *Aerospace America*, November 1994