DEPARTMENT OF THE AIR FORCE
Software Technology Support Center

Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems Command and Control Systems Management Information Systems

Condensed Version
February 2003

Preface

The U.S. Air Force's Software Technology Support Center is excited to provide an updated and condensed version of the *Guidelines for Successful Acquisition and Management of Software Intensive Systems (GSAM)*.

We are pleased that prior editions have been so well received and that many individuals and programs have worked hard to implement the principles contained therein. These users also repeatedly requested a more streamlined version of the content. Our goal for this project has been to provide a usable desk reference that would give a brief but effective overview of important software acquisition and development topics, provide checklists for rapid self-inspection, and provide pointers to additional information on the topics covered.

As we assembled these *Guidelines* using the experiences of software acquisitions and software development projects, assessments and consulting, we have found a handful of key principles that, when not followed, cause projects to suffer. Although these principles seem so basic, we have noticed that they are often neglected or are not implemented because they run counter to the prevailing environment:

- We must focus on the true customer.
- We must spend more energy communicating with this customer and working towards a quality product for the program. We must minimize, as much as possible, the time spent talking about the encompassing politics but focus instead on innovation, collaboration, and flexibility.
- We must understand the importance of the full lifecycle of our program and the accompanying product or service.
- We must baseline our requirements and project scope as soon as possible.
- We should break up our program into smaller phases or multiple projects, if necessary, to gain the advantage of incremental success.
- We must introduce measurements into our programs and appropriately use them for better predictability of costs, schedule, and quality and management of the program as it progresses. These measurements being used to also stimulate increased accountability into our cultures.
- We must not be afraid to talk about the risks associated with our programs and projects, and focusing on, tracking, and managing the key risks.
- We must capture relevant data, lessons learned, and other historical information from our programs with a mantra of organizational learning, regardless of the potential and actual staff and management turnover.
- We must work towards and emphasize sponsorship in our improvement efforts, where leaders and managers understand these principles and "walk the talk."

We hope you find this reference useful. Note: If you notice errors or have a suggestion for improvement, please provide your recommendations to us at:

Customer Service
OO-ALC/MASE
6022 Fir Avenue
Hill AFB, Utah 84056-5820
801-775-5555
www.stsc.hill.af.mil

Larry W. Smith
Project Manager
Software Process Improvement
Software Technology Support Center

Condensed GSAM Handbook

CONTENTS

This page intentionally left blank.

Chapter 1

Overview of Project Management

CONTENTS

This page intentionally left blank.

Chapter 1

Overview of Project Management

1.1 Introduction

This chapter provides a brief overview of project management, its purpose, activities, and responsibilities. The detailed activities will be covered in subsequent chapters. This material has been condensed from multiple sources. The sources are listed along with other recommended web resources in Section 1.8, Resources, and provide more detail and direction for managing projects. Check them out!

The following sections will discuss what projects are, what project management is, and what project management generally entails. Next is a summary of project life cycles and their phases, along with the processes and activities of project management. The chapter concludes with checklists, definitions, and further resources.

1.1.1 Projects and Programs

A project is a group of activities undertaken to meet one or more specific objectives. These objectives could include solving a problem, building or upgrading a system or product, launching a product or service, implementing a strategic plan, changing a process, or one of many other unique efforts.

Projects can differ in size from small and simple to large and complex. However, to accomplish specific objectives, projects are temporary and have specific starting and completion dates. Ongoing operations such as operating a maintenance facility or publishing a magazine are not projects. Performing a specific aircraft avionics upgrade or printing a monthly issue of a magazine are projects. Limited, specific performance times and objectives are how projects differ from programs. Programs are generally much larger efforts than projects, with longer duration. Relative to projects, they are ongoing rather than temporary efforts. While this chapter focuses primarily on project management, most of what is presented will also apply to programs. Programs are made up of multiple projects and in many cases can be treated as longer, more complex projects.

Projects are often divided into smaller components or activities, usually based on technical and functional disciplines such as engineering, manufacturing, testing, and procurement. The relationships between programs, projects, and activities are shown in Figure 1-1. Some projects are divided along product lines instead of activities.

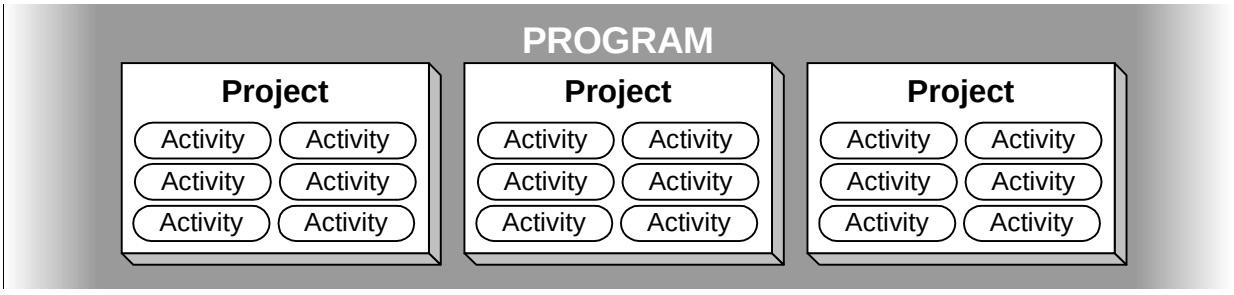

Figure 1-1 Relationships Between Programs, Projects, and Activities.

Projects are successfully complete when their objectives have been achieved. Projects should be terminated when it can be seen they will fail to meet their objectives.

1.1.2 Project Management

Successful projects rarely happen by themselves. They must be planned and executed. They must have support from management and from the organization in general. To be successful projects must also have a responsible and empowered manager to drive, direct, and monitor the project. Project management is that discipline which employs skills and knowledge to achieve project goals through various project activities. It involves controlling costs, time, risks, project scope, and quality through project management processes.

The functions of project management include:

- Planning – Planning the project and establishing its lifecycle.
- Organizing – Organizing resources: personnel, equipment, materials, facilities, and finances. Coordinating work and resources.
- Leading – Assigning the right people to the right job. Motivating people. Setting the course and goals for the project.
- Controlling – Evaluating progress of project and, when necessary, applying changes to get it back on track.

Performing these functions in an organized framework of processes is the job of the Project Manager.

1.1.3 Project Manager

The selection of a Project Manger (PM) has a major effect on the success of the project. The PM should have the skill, knowledge, and personality necessary to bring the project to fruition. In addition to these traits, the PM must be given the level of responsibility and authority necessary to perform the job.

The actual role of the project manager depends on the structure of the organization in which he or she works. Organizations can be function-oriented, project-oriented, or some type of matrix in between. In a heavily project-oriented organization the PM may have relatively unlimited authority, answering only to upper management. At the other end of the spectrum is an organization that manages by function. The PM must deal with functional managers as equals, or possibly even superiors, and negotiate for resources. Most organizations fall somewhere in between these two extremes. Figure 1-2 depicts the level of PM authority associated with different types of organizations. It is essential that the PM understands the workings of the organization and knows the level of authority that goes with the job. It is also essential that upper management grant authority and establish an environment which will enable the PM to successfully accomplish the project objectives

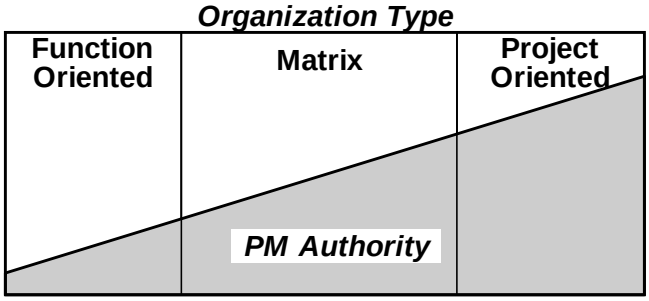

Figure 1-2 Organization Type and PM Authority

Project Managers need both management and technical skills. The key management skills are those needed to perform or direct project management activities, and are listed in Table 1-1.

Table 1-1 Management Skills for Project Managers

Skill	Description
Integration Management	Coordinate development of the Project Plan, execution of the Plan, and the change control process to ensure all aspects of the project are working together.
Scope Management	Establish scope of the project at the start. Develop and implement plans and procedures to verify that scope is achieved and maintained. Define and oversee the proc-

Skill	Description
	ess for controlling changes to the scope.
Risk Management	Identify potential risks. Mitigate large risks and plan how to deal with smaller risks. Monitor the project to detect and resolve problems.
Time Management	Estimate the duration of project activities, sequence activities, and develop and control the project schedule.
Cost Management	Estimate project costs and develop and control the project budget.
Quality Management	Establish and control processes to ensure project goals are met to the satisfaction of the stakeholders. This includes quality planning, quality assurance, and quality control.
Communications	Define methods and lines of reporting and information distribution. Who gets reports and project information, how often, and what is the content?
Procurement Management	Oversee procurement and delivery of materials, equipment, and services needed for the project. Includes planning, solicitation, source selection, and contract administration.
Human Resources Management	Develop good leadership qualities. Plan team organization, obtain the right people to staff the positions, and develop their skills as individuals and as a team.
Earned Value Management	A Systematic approach to project control integrating cost and schedule control with performance control. (See Chapters 6 and 7.)

The Project Manager's technical skills should include at least some technical understanding in the project field. Remember, however, the PM will not be doing the technical work but is to direct the work done by others. The essential level of expertise is the ability to understand what is being done by others, not necessarily how.

1.2 Process Description

To understand how a project is brought to fruition, the PM must understand project processes. The PM must also understand the differences between program and project life cycles. It is also important to understand the difference between life cycle phases and project management processes.

1.2.1 Program/Product Life Cycle

As stated earlier, programs are generally large efforts spanning long periods of time and are composed of multiple projects. They are usually associated with developing or acquiring systems such as aircraft, weapons, training operations, communications, etc. An example product life cycle with its associated phases and their products is shown in Figure 1-3.

This example has five phases, Planning through Operation, each of which produces a specific output, including a satisfied user for the last phase. At the end of each phase a decision is made to continue or not to the next phase. When the product is completed, it is implemented, and after being in operation for some length of time, it is retired. Various industries employ different life cycles, depending on their products. Each phase of a product life cycle can consist of one or more projects.

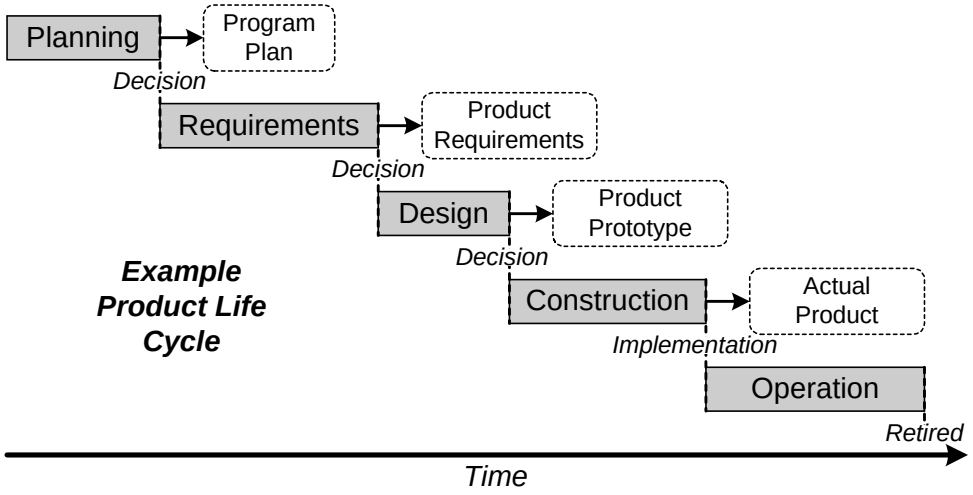

Figure 1-3 Example Product Life Cycle

1.2.2 Project Life Cycle

Projects, like programs, have life cycles and are usually performed in phases. Each phase accomplishes specific work toward reaching the project goal and produces one or more deliverables, depicted in Figure 1-4. These are tangible, real items used in attaining the final goal of the project, and could include plans, studies, designs, or software or hardware prototypes. The end of a phase is defined by completing its deliverable.

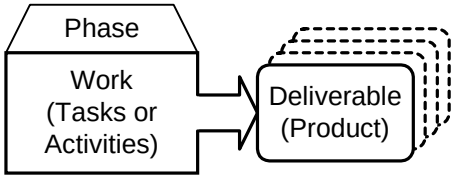

Figure 1-4 Phases Produce Deliverables

Figure 1-5 illustrates an example project life cycle, with its phases and their major deliverables.

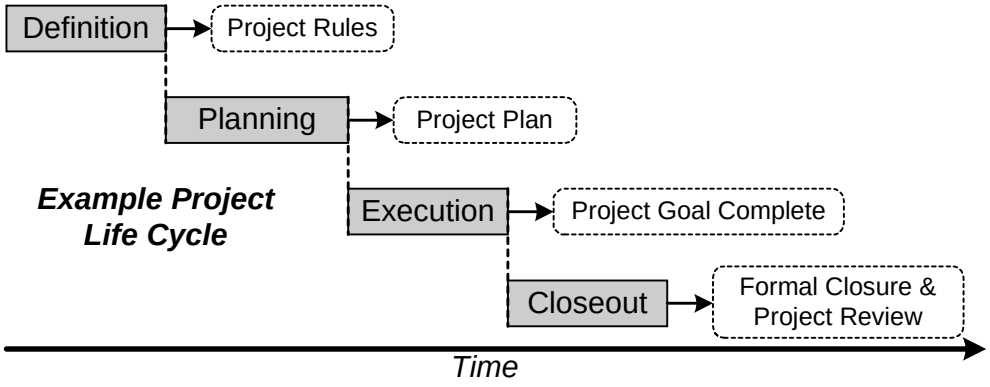

Figure 1-5 Example Project Life Cycle

While major aspects of project management are applicable across all projects, life cycles may vary, depending on the type of project and the organization performing the work. It is important to implement an appropriate life cycle for the product. Chapter 2 of this book introduces and discusses various software development life cycles. This chapter deals with a generic project life cycle.

1.2.3 Project Phases

The phases identified in Figure 1-5 are common across most projects. However, they may be called by different names or split into additional phases. They may even be iterative, where, for example, a prototype is designed, built,

and tested, then the results are used to design, build, and test a new prototype. Project phases should in most cases be comparable to the generic project phases discussed here.

1.2.3.1 Definition Phase

This phase begins when a Project Charter is created by upper management that defines the project's purpose and identifies a project manager. The Charter should also include a statement of support authorizing the PM to perform his or her functions. During this phase, the rules of the project are defined. The PM and stakeholders determine the goals, scope, and constraints of the project. Key individuals and groups are identified as members of the project core team and their roles are defined by the PM and upper management. Communications channels, authority, and the chain of command are also defined by upper management with the PM. These project rules are written in three documents, the Project Statement of Work (PSOW), the Project Responsibility Matrix, and the Project Communication Plan. The PSOW establishes the scope of the project and documents what is to be accomplished. For an internal project the PSOW becomes the primary requirements document. However, the PSOW is not the same as a contract Statement of Work (SOW). For a project where much of the work is contracted, the SOW is binding, contractual agreement. These documents are described in Section 1.6, Definition of Terms. Figure 1-6 depicts the input, major activities, and products of the Definition Phase.

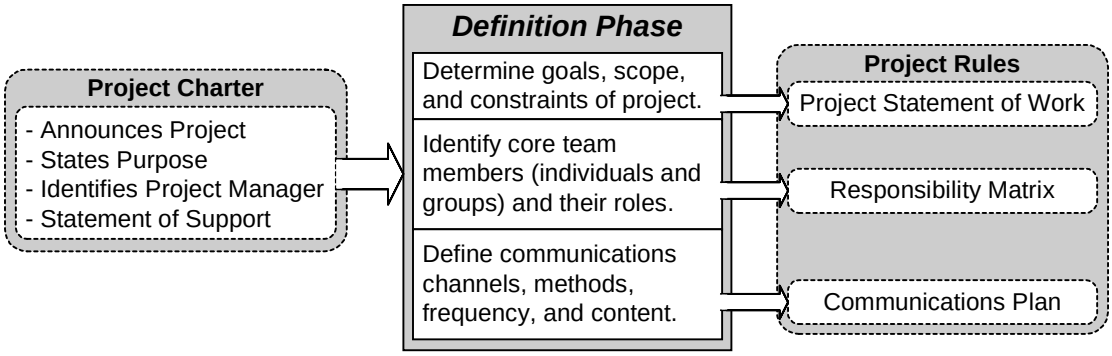

Figure 1-6 Project Definition Phase

1.2.3.2 Planning Phase

The planning phase uses the Project Rules as a foundation and defines the path to achieve the project goals. It is performed by the PM and the core project team, interfacing with appropriate elements of the organization, and identifies the actual work to be done. It includes estimating time, cost, and resources required to perform the work, and produces plans to serve as a baseline and direct the work. A key part of schedule planning is identifying the *critical path*. This is the chain of interdependent, sequential project activities which takes the longest time to complete, and thus determines the minimum schedule for the project. Planning also includes risk identification and risk reduction efforts. The results of the Planning Phase become the Project Plan.

Figure 1-7 shows the inputs, activities, and products of the Planning Phase. Note the feedback loop from the phase activities to the Project Rules. This indicates that the rules may need to be modified after more detailed analysis in this phase reveal deficiencies or inefficiencies in the rules. This illustrates the iterative nature of project management. Remember the Project Plan is fluid and the PM should expect changes.

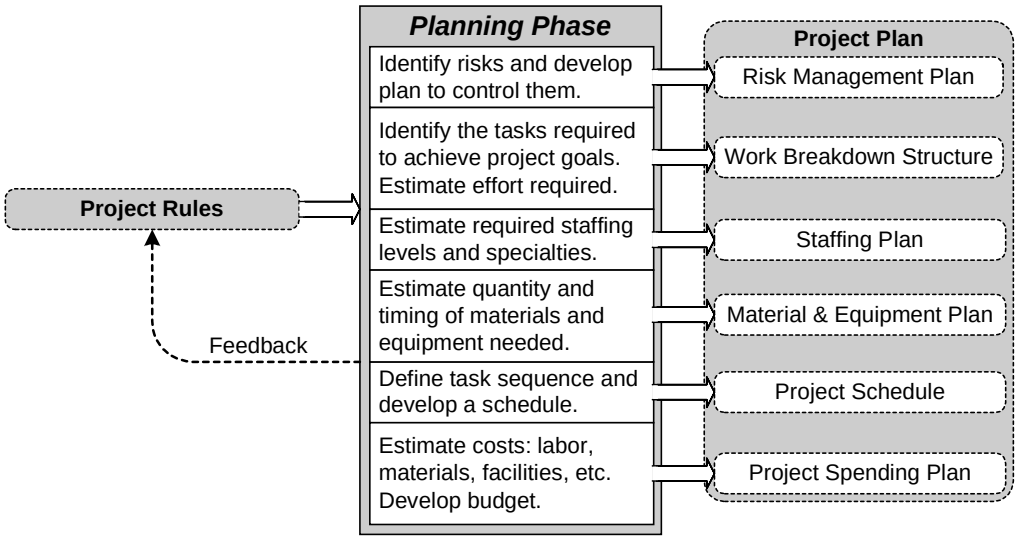

Figure 1-7 Project Planning Phase

1.2.3.3 Execution Phase

With a Project Plan for guidance, the actual project work can begin in earnest. This is the phase where the project goals are achieved. While Figure 1-8 may make it look far simpler than the Planning Phase, the Execution Phase entails directing the various work groups in their activities, monitoring their progress, solving problems and resolving issues that will certainly come up, making changes to the plan, and coordinating these changes. These activities are part of the executing and controlling processes discussed in Section 1.2.4. If your planning has been done well, you will have a smoother ride through this phase. This phase is complete when the product is complete or the project goals are reached.

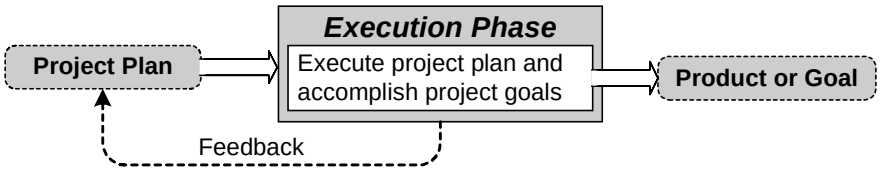

Figure 1-8 Project Execution Phase

1.2.3.4 Closeout Phase

The Closeout Phase begins with the delivery of the product or completion of the project goals. It consists primarily of tying up loose ends. Any unresolved issues from the contract or Statement of Work are resolved in this phase. The contract is signed off as fulfilled and all other paperwork is completed. A very important activity of this phase is assembling the project history. This is a summary of all that has been accomplished. It should include information that will allow you or a follow-on project manager to understand what was done, and why. Of particular importance is a compilation of lessons learned from the project so you or others in your organization can do things better on the next project. Figure 1-9 summarizes the Closeout Phase.

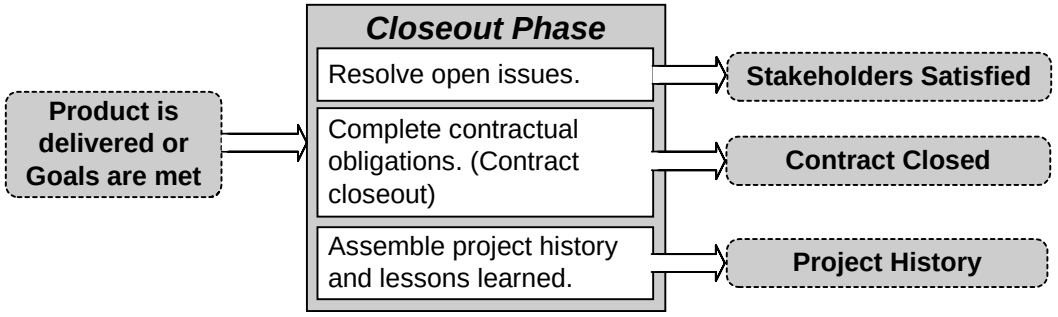

Figure 1-9 Project Closeout Phase

1.2.4 Project Processes

The Program Management Institute (PMI) defines five major process groups used in projects. Processes are sequences of activities that accomplish specific functions necessary to complete or enable some portion of the project. These are not phases themselves but can be found both in projects and in each major phase of a program or large project. Because the activities in later phases may require changes in the products of earlier phases, these processes become iterative and often overlap phases as well as each other. An example of this would be an issue in the Execution Phase requiring a change to plans made in the Planning Phase. This overlap is shown in Figure 1-10.

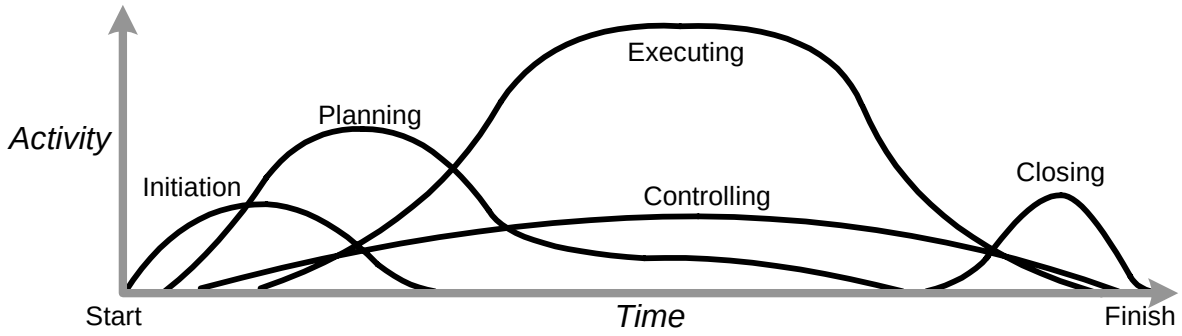

Figure 1-10 Project Management Processes Overlap, per PMI

1.2.4.1 Initiation Process

The initiation process consists of formally validating or authorizing the project. It often includes some form of analysis, such as a feasibility study, a preliminary requirements study, a concept of operations, or a preliminary plan.

1.2.4.2 Planning Processes

Planning processes establish the scope or boundaries of the project. They lay the foundation and define an expectation baseline. Future proposed changes are evaluated against this baseline. What must be balanced here and throughout the project are schedule, cost, and quality. Changes to the scope of the project will almost certainly affect at least one of these, requiring changes in the others to achieve balance again. Likewise, changes in one or more of these three constraints will require changes in the others and/or changes to the scope or expectations of the project. This balance is shown in Figure 1-11. Note that these do not necessarily define scope but they do constrain it.

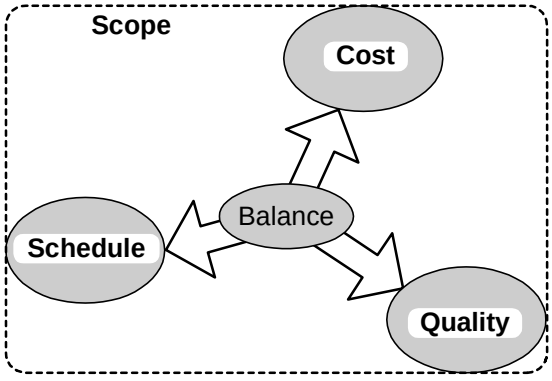

Figure 1-11 Balancing Constraints Within Project Scope

Other planning processes include the following:

- Definition of activities needed to perform project
- Estimating activity duration
- Development of a schedule
- Risk management
- Communications planning
- Staff planning
- Organization definition
- Sequencing of activities
- Resource planning
- Estimating costs
- Developing a spending plan or budget
- Quality planning
- Procurement planning
- Developing a Project Plan

1.2.4.3 Executing Processes

The executing processes are those that direct or enable the actual work of the project. They consist of the following:

- Executing the Project Plan.
- Quality assurance activities.
- Procurement activities.
- Developing team and individual competencies.
- Communicating to team members and stakeholders.

1.2.4.4 Controlling Processes

Controlling processes are ongoing throughout most of the project. They include verifying that the project is proceeding according to plan or determining where and how much a deviation is occurring. They are absolutely essential to the progress and success of the project. They include:

- Monitoring, measuring, and reporting the performance of project activities.
- Verifying the project is continuing within scope.
- Controlling changes to the project scope.

These processes are in-turn enabled by these supporting processes:

- Schedule Control
- Cost Control
- Quality Control
- Risk Monitoring and Control.

1.2.4.5 Closing Processes

The closing processes are accomplished following the completion of the project objectives. Their purpose is to resolve any open issues, complete any paperwork required for formal completion of the project, and gather information useful for evaluating project performance for future reference. The first process is Contract Closeout, where any remaining contract issues are settled. The other process is Administrative Closure, where formal documents terminating the project are generated, and an appropriate history of project performance and lessons learned is gathered.

1.3 Project Management Application

The application of the previous information to a real project will depend on several things. A PM assigned to an ongoing project has little control over how the project is set up. In this case the new PM will need to quickly learn the following:

- Project purpose and objectives
- Major risks

- Project phases and their deliverables
- Project budget and current spending status
- Project schedule and current status
- Current problems and issues
- Project team organization and contacts
- Project management processes in place or planned
- Life cycle of the product the project is supporting
- Communications – who gets what information when

A new project requires the PM to learn or establish the items in the previous list. After understanding the purpose and goals of the project, the PM will need to select an appropriate project life cycle. If it is a small, straightforward software development effort, all the software development life cycle phases are performed as part of the execution phase, as shown in Figure 1-12. Remember to distinguish between project phases and software development phases. Also note that this example portrays only one of several possible life cycle models.

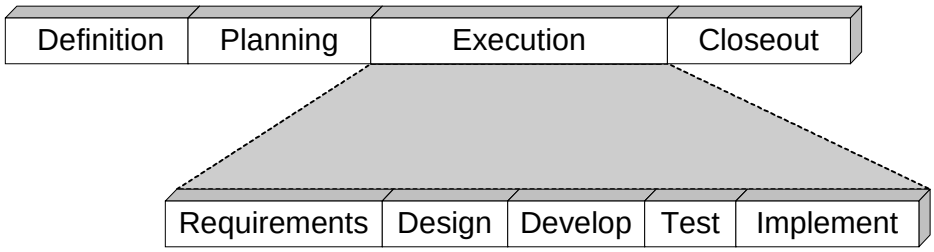

Figure 1-12 Software Development Phases Are Part of Project Execution Phase

If the software development effort is larger or more complex, the development life cycle will still be performed in the execution phase of the overall project or program. However, each phase of software development now becomes a project in its own right, with all the phases of an individual project. This is shown in Figure 1-13.

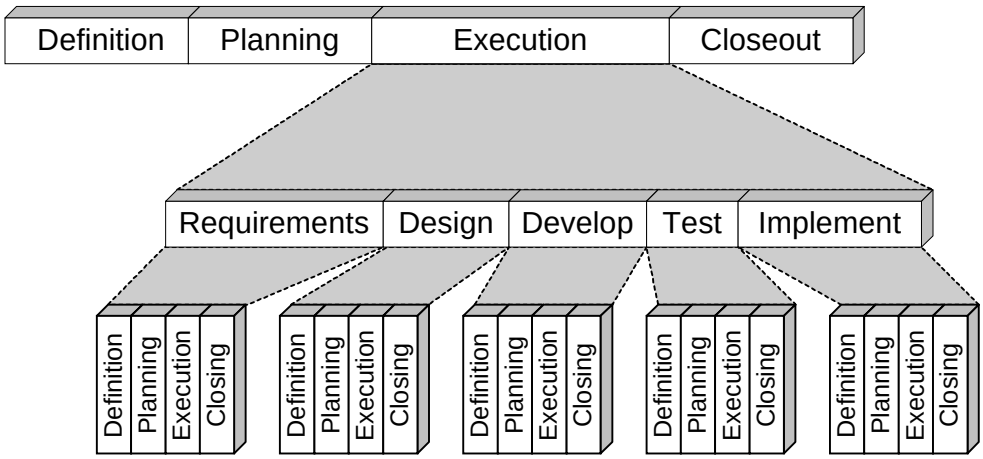

Figure 1-13 Complex Development Phases Become Projects Themselves

If the PM is managing a project team developing software, then he or she will manage both project and software development phases. If managing a contract effort, the PM will manage the overall project, but the actual development effort will be managed by a contractor PM.

With a contracted software development effort, you will also need to add a procurement phase to your project. This additional phase will take the outputs from the previous stage, including a Statement of Work, and perform procurement activities to contract with an outside organization to perform the work. This additional phase is shown in Figure 1-14.

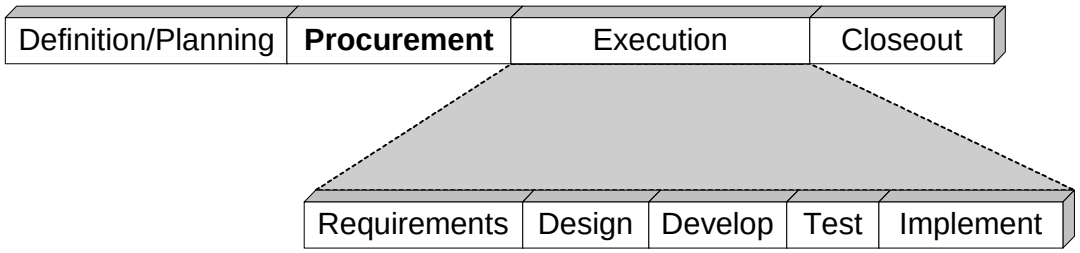

Figure 1-14 Project With Procurement Phase

Knowing the project goals and selecting a project life cycle establish the foundation on which to build the project.

1.4 Project Management Checklist

This checklist is provided to guide you in essential actions to ensure your project is on track in meeting cost, schedule, and performance requirements. If you cannot check an item off as affirmative, you need to either rectify the situation or develop a contingency plan to solve problems that may arise. For example, if the staff does not have sufficient technical skill to do the work, you will need to remedy the situation by providing training, or by obtaining sufficiently skilled people.

1.4.1 Beginning a Project

- ❑ The project has specific goals to accomplish and you understand the reasoning behind them.
- ❑ All stakeholders (interested parties) understand and agree on the expected project outcomes.
- ❑ Upper management is solidly behind the project.
- ❑ You understand the level of authority you have been granted in relation to the project and the rest of the organization and the level of authority is appropriate.
- ❑ You understand how the organization operates, including how to get things done within the organization.
- ❑ You understand what you are responsible for delivering at both a macro and a micro level.
- ❑ You know the high-priority risks your project faces.

1.4.2 During Project Planning

- ❑ You know which external interfaces are not under your control.
- ❑ You know the estimated size of the software to be developed, and how the estimate was made.
- ❑ Funding has been allocated for the project.
- ❑ A credible budget has been prepared, based on project scope and work estimates.
- ❑ Adequate time has been allocated to complete the project.
- ❑ Adequate staff is or will be available to complete project tasks.
- ❑ The project staff has sufficient expertise to perform the work.
- ❑ Facilities and tools are or will be available for the project team.
- ❑ You know of potential funding cuts and when they might come.
- ❑ You know what major problems have plagued projects of this type in the past.
- ❑ An appropriate life cycle has been selected for the project and you understand that life cycle.

- ❑ You have a credible Work Breakdown Structure (WBS).
- ❑ All requirements have work tasks assigned to fulfill them.
- ❑ All work tasks are associated with project requirements or support activities.
- ❑ Special requirements or constraints are documented.
- ❑ You have a budget, schedule, and performance baseline established and documented.
- ❑ You have identified the critical path for the project.
- ❑ You have a process established to monitor the project and detect problems and departures from the baseline.

1.4.3 During Project Execution

- ❑ You know what your project's expenditures are to-date and any difference between that and your budget.
- ❑ You know the status of project activity completion along the critical path and any difference between that and the schedule.
- ❑ You are aware of any issues or problems with quality or performance that may impact the critical path.
- ❑ You are aware of any contract performance issues.

1.5 Regulations

Clinger-Cohen Act of 1996, The National Defense Authorization Act for Fiscal Year 1996.

DoD Regulation 5000.2-R, Mandatory Procedures for Major Defense Acquisition Programs (MDAPS) and Major Automated Information System (MAIS) Acquisition Programs, 10 June 2001, Part 2; 2.8 Support Strategy, Part 2; 2.9 Business Strategy and Part 5; Program Design.

DoDD 5000.1, The Defense Acquisition System, 4.5 Effective Management.

DoDI 5000.2, *Operation of the Defense Acquisition System*, 4.7 The Defense Acquisition Management Framework.

FAR -- Part 39; *Acquisition of Information Technology*; (FAC 97-27); 25 June 2001.

OMB Circular A-130; *Management of Federal Information Resources*; Revised -- February 8, 1996.

1.6 Definition of Terms

Baseline – A standard against which future status, progress, and changes are compared and measured. Most plans developed during the Planning Phase are used as baselines. The budget usually serves as one baseline, the schedule as another, etc.

Communications Plan – Document that defines the lines, content, method, and frequency of communications between the project manager, members of the project team, stakeholders, and management.

Critical path – The sequence or chain of interdependent activities in the project that takes the longest time to complete. This sequence determines the shortest schedule for the project. Any delay in a critical path activity increases the project schedule.

Life Cycle – The complete set of phases something goes through, beginning with its conception and ending with its retirement from service.

Process – A series of related activities or steps that accomplish a specific purpose.

Project Charter – Document that announces the project by name, states its purpose, identifies the project manger, and announces his or her authority.

Project Manger (PM) – Individual with responsibility and authority for directing the project.

Project Statement of Work (PSOW) – Document that defines the goals, scope, and constraints of the project. It states what needs to be done, not how to do it.

Responsibility Matrix – Document that identifies members of the project team and defines their roles.

Stakeholders – Those persons and organizations that have an interest in the performance and completion of the project. The customer or user of a product created through a project is usually a primary stakeholder.

Statement of Work (SOW) – A contractual document that defines the work to be performed for a specific project under contract.

Work Breakdown Structure (WBS) – A breakdown of the project into its constituent tasks or activities. It lists the specific work needed to complete all aspects of the project.

1.7 Resources

AllPM - Project Managers Homepage. www.allpm.com

Best Manufacturing Practices (BMP), TRIMS Risk Management and Best Practices software downloads: www.bmpcoe.org/pmws/index.html

Best Manufacturing Practices (BMP) Library. Download KnowHow software and copies of DOD 5000.1, 5000.2, etc: www.bmpcoe.org/pmws/download/knowhow.html

Can-Plan project management software download: www.geocities.com/billmcmillan2000/CAN-PLAN.html

Complete Idiot's Guide to Project Management, Second Edition. Baker, Sunny and Kim. USA: Alpha Books, 2000.

Defense Acquisition Deskbook:. http://web2.deskbook.osd.mil/default.asp?

Defense Acquisition Deskbook, Program Management: http://web1.deskbook.osd.mil/CS_PM.asp

Defense Systems Management College. Back issues of *Program Manager Magazine.* www.dsmc.dsm.mil

DoD Software Clearing House: www.dacs.dtic.mil

Gantthead Online Community for IT Project Managers: www.gantthead.com

Guide to the Project Management Body of Knowledge, A, 2000 Edition, Project Management Institute. USA: John Wiley & Sons, Inc., 1999.

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, OO-ALC/TISE, May 2000. Available for download at: www.stsc.hill.af.mil/gsam/guid.asp

Project Management Forum: www.pmforum.org

Project Management Institute (PMI): www.pmi.org

Project Management Knowledge Base. Extensive free library. www.4pm.com

Project Manager Magazine, Defense Systems Management College. Subscribe at: www.dau.mil/forms/order_pm.asp

Software Program Managers Network (SPMN). Sponsored by the Deputy Under Secretary of Defense for Science and Technology (DUSD (S&T), Software Intensive Systems Directorate: www.spmn.com

SPMN Risk Radar software download: www.spmn.com/rsktrkr.html

SPMN Guidebooks available for download at: www.spmn.com/products_guidebooks.html

Software Technology Support Center (STSC), OO-ALC/TISE: www.stsc.hill.af.mil

TechRepublic Information Technology Forum: www.techrepublic.com

Ten Step Project Management Process Site: www.tenstep.com

Chapter 2

Software Life Cycle

CONTENTS

This page intentionally left blank.

Chapter 2

Software Life Cycle

2.1 Introduction

In the early days of computing, software was developed by many individuals following their own methods. Often, the methods employed some form of "code and fix", where the programmer writes some code and then tests it to see how it performs. The programmer then uses the test results to modify or fix the code and tests again. Programmers were able to get by with this type of development for two reasons. First, no better way had been developed, and second, software was not that complex. As software grew more complicated and organizations relied on computers for more of their operations, including finances and even human lives, this laissez faire approach to programming gave way to more disciplined methods. The overall framework in which software is conceived, developed, and maintained is known as the Software Development Life Cycle (SDLC). This chapter discusses the various types of SDLCs, along with their advantages and disadvantages.

A life cycle model defines the phases, milestones, deliverables, and evaluation criteria of the software development process. These form the basis of the work breakdown structure (WBS), used for project planning and management.

2.2 Process Description

Life cycles are usually referred to as models, and define the phases of a software development effort. Simple life cycles may have only three phases, Design, Development, and Maintenance; while complex life cycles may include 20 or more phases. Generally, software life cycles include the phases shown in Figure 2-1.

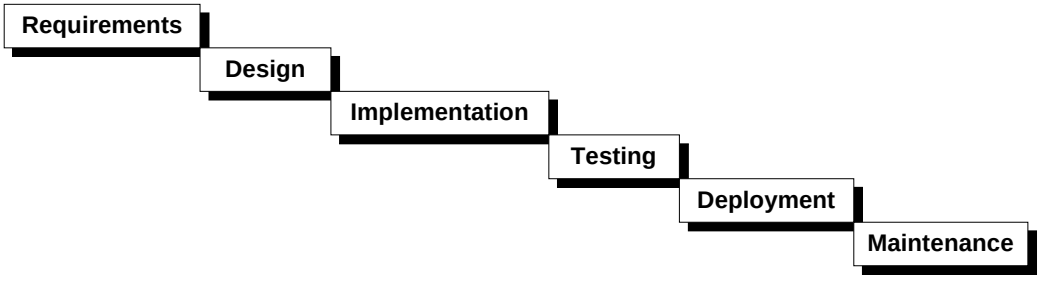

Figure 2-1 Common Life Cycle Phases

These "classic" phases are often divided into additional phases to allow better definition and control of the development process. They may also be repeated in an iterative manner, depending on the software complexity and the life cycle model used. Most life cycle phases are identical or similar to the common phases identified above and the following general descriptions will apply across most models. Note that single phases are composed of multiple activities.

The *Requirements Phas*e consists of analyzing the problem or need for which the software is being developed. This analysis, a systems engineering activity, develops and specifies requirements, stating what the software must do. In addition to stated requirements, requirements are derived from higher-level requirements and statements of need.

In the *Design Phase* the software structure is defined. Technical approaches are chosen and problems are solved conceptually. This phase is often divided into a Preliminary Design Phase and a Detailed Design Phase. In the preliminary design the initial software architecture is developed. In the detailed design, functional modules are defined, along with user interfaces and interfaces between modules.

The *Implementation Phase* (sometimes called the Development Phase) is where the programming or coding takes place to execute the software design. This phase is often iterative, with unit and integration testing being performed after a software build, and the results used in another round of programming.

Software is tested for functionality and requirements compliance during the *Testing Phase*. Testing is often split into three separate phases: Unit Testing, Integration Testing, and Acceptance Testing. The first two may be part of a repeated cycle of coding and testing, while acceptance testing verifies requirements compliance.

During the *Deployment Phase* the software is installed in the intended system and users are trained in its operation. At this point the software development effort is considered complete.

The *Maintenance Phase* includes fixing errors and modifying or upgrading the software to provide additional functionality, such as enabling the software to work with new computing platforms. This phase costs far more in time and effort than the original development. Software maintainers must relearn the original software code to understand what was done, then make changes to specific modules to produce the desired effect without interfering with rest of the software. It's much easier to change requirements earlier than it is to change software code later. This also means that software should be developed with maintenance in mind.

2.2.1 Life cycle Models

There are many life cycle models, or paradigms. Most of them are variations of three classic software development models: the waterfall, incremental, and spiral models. These three, along with the evolutionary model, are summarized here.

2.2.1.1 Waterfall Model

The waterfall model, also known as the linear sequential model, is shown in Figure 2-2 with its major phases, milestones, and products. It is a highly structured development process, first used on DoD software projects in the 1970s. It is the "traditional" approach to software development and was derived from defense and aerospace project lifecycles. It is considered superior to the previously used "code and fix" methods of software development, which lacked formal analysis and design.

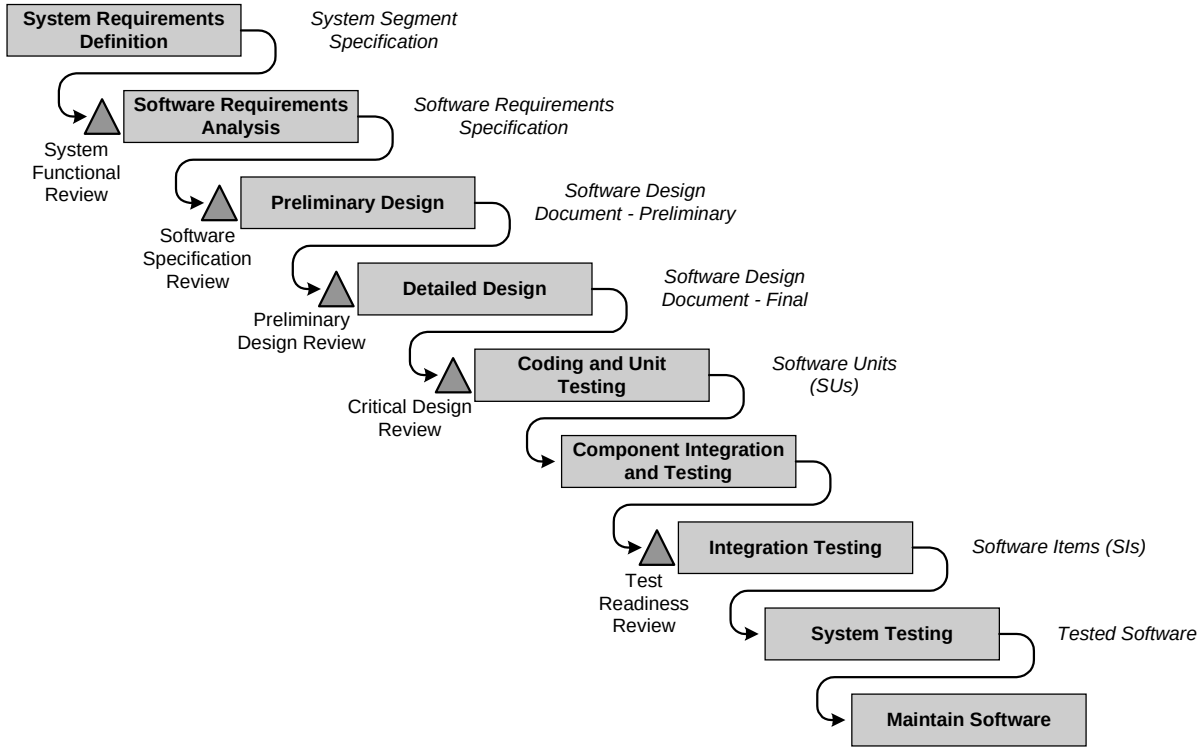

Figure 2-2 Waterfall Model [1]

The waterfall model is documentation-intensive, with earlier phases documenting *what* must be done and subsequent phases adding greater detail and defining *how* it should be done. The output from one phase serves as the input to the next phase, with the project flowing from one step to the next in a waterfall fashion. Phases are assumed to be sequential, with only localized feedback during the transition between phases. This is accomplished by using reviews as gates. Comprehensive reviews validate the work of one phase, and require the resolution of any problems before development is allowed to proceed to the next phase.

An important consideration for the Waterfall model is that fixes or modifications are often put off until the maintenance phase. This can be very costly, as the cost to correct a problem gets higher with each successive phase.

Advantages

- System is well documented.
- Phases correspond with project management phases.
- Cost and schedule estimates may be lower and more accurate.
- Details can be addressed with more engineering effort if software is large or complex.

Disadvantages

- All risks must be dealt with in a single software development effort.
- Because the model is sequential, there is only local feedback at the transition between phases.
- A working product is not available until late in the project.
- Progress and success are not observable until the later stages. If a mistake or deficiency exists in the documentation of earlier phases, it may not be discovered until the product is delivered.
- Corrections must often wait for the maintenance phase.

Application

The Waterfall model can be successfully used when requirements are well understood in the beginning and are not expected to change or evolve over the life of the project. Project risks should be relatively low.

2.2.1.2 Incremental Model

The incremental model is essentially a series of waterfall cycles. One variant is shown in Figure 2-3. The requirements are known at the beginning of the project and are divided into groups for incremental development. A core set of functions is identified in the first cycle and is built and deployed as the first release. The software development cycle is repeated, with each release adding more functionality until all requirements are met. Each development cycle acts as the maintenance phase for the previous software release. While new requirements that are discovered during the development of a given cycle can be implemented in subsequent cycles, this model assumes that most requirements are known up front. The effort is planned and executed to satisfy the initial list of requirements. A modification to the incremental model allows development cycles to overlap. That is, a subsequent cycle may begin before the previous cycle is complete.

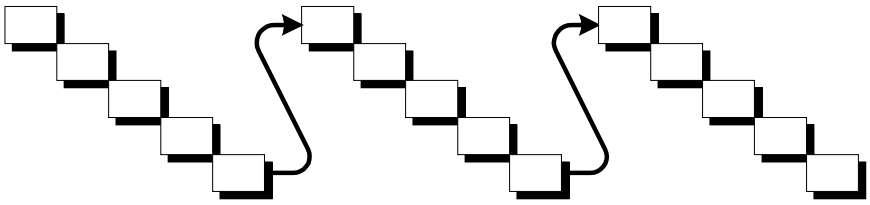

Figure 2-3 The Incremental Model is a Series of Waterfalls

Advantages

- Provides some feedback, allowing later development cycles to learn from previous cycles.
- Requirements are relatively stable and may be better understood with each increment.

- Allows some requirements modification and may allow the addition of new requirements.
- It is more responsive to user needs than the waterfall model.
- A usable product is available with the first release, and each cycle results in greater functionality.
- The project can be stopped any time after the first cycle and leave a working product.
- Risk is spread out over multiple cycles.
- This method can usually be performed with fewer people than the waterfall model.
- Return on investment is visible earlier in the project. [7]
- Project management may be easier for smaller, incremental projects. [7]
- Testing may be easier on smaller portions of the system.

Disadvantages

- The majority of requirements must be known in the beginning.
- Formal reviews may be more difficult to implement on incremental releases than on a complete system. [2]
- Because development is spread out over multiple iterations, interfaces between modules must be well-defined in the beginning. [2]
- Cost and schedule overruns may result in an unfinished system.
- Operations are impacted as each new release is deployed.
- Users are required to learn how to use a new system with each deployment.

Application

The incremental model is good for projects where requirements are known at the beginning, but which need functionality early in the project or which can benefit from the feedback of earlier cycles. Because each cycle produces a working system, it may also be advantageous for projects whose continued funding is not assured and may be cut at any time. It is best used on low to medium-risk programs. If the risks are too high to build a successful system using a single waterfall cycle, spreading the development out over multiple cycles may lower the risks to a more manageable level. [3]

2.2.1.3 Evolutionary Model (Prototyping)

The evolutionary model, like the incremental model, develops a product in multiple cycles. Unlike the incremental model, which simply adds more functionality with each cycle, this model produces a more refined prototype system with each iteration. The process, shown in Figure 2-4, begins in the center with initial requirements and plans, and progresses through multiple cycles of planning, risk analysis, engineering, and customer evaluation. Each cycle produces a prototype that the customer evaluates, followed by a refinement of requirements.

Specification, development, and testing activities are carried out concurrently (in the engineering quadrant) with rapid feedback. Since requirements continue to change, documentation is minimal, although essential information must still be included for understanding the system and for future support. Implementation compromises are often made in order to get the prototype working – permanent fixes can be made with the next prototype. Operational capability is achieved early, but users must be willing to learn how to use each new prototype.

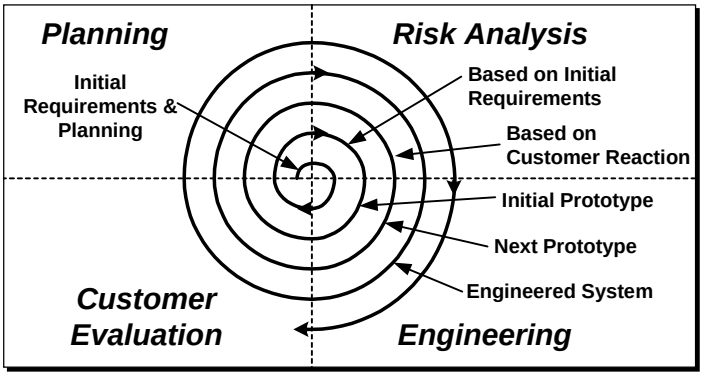

Figure 2-4 First Generation Evolutionary Model [4]

General system requirements must be known prior to development. This is particularly helpful where evolving technology is being intro-

duced into the project. The evolutionary model relies heavily on user feedback after each implementation to refine requirements for the next evolutionary step.

Advantages [5]

- Project can begin without fully defining or understanding requirements.
- Final requirements are improved and more in line with real user needs.
- Risks are spread over multiple software builds and controlled better.
- Operational capability is achieved earlier in the program.
- Newer technology can be incorporated into the system as it becomes available during later prototypes.
- Documentation emphasizes the final product instead of the evolution of the product. [2]
- This method combines a formal specification with an operational prototype. [2]

Disadvantages [5]

- Because there are more activities and changes, there is usually an increase in both cost and schedule over the waterfall method.
- Management activities are increased.
- Instead of a single switch over to a new system, there is an ongoing impact to current operations.
- Configuration management activities are increased.
- Greater coordination of resources is required.
- Users sometimes mistake a prototype for the final system.
- Prototypes change between cycles, adding a learning curve for developers and users.
- Risks may be increased in the following areas:
  - Requirements – Temptation to defer requirements definition.
  - Management – Programs are more difficult to control. Better government/contractor cooperation needed.
  - Approval – Vulnerable to delays in funding approval, which can increase schedule and costs.
  - Architectural – Initial architecture must accommodate later changes.
  - Short term benefits – Risk of becoming driven by operational needs rather than program goals.
  - Risk avoidance – Tendency to defer riskier features until later.
  - Exploitation by suppliers – Government bargaining power may be reduced because initial contract may not complete the entire task, and subsequent contracts are not likely to be competed.
  - Patchwork quilt effects – If changes are poorly controlled, the product quality can be compromised.

Application

The evolutionary model can be employed on most types of acquisitions. However, it is usually employed on medium to high-risk systems. The evolutionary model should be considered for systems where requirements are not all known or not yet refined, but are expected to evolve. It is more applicable to new systems than upgrading existing software. [2] The developing and using organizations must be flexible and willing to work with evolving prototypes. Programs well suited to employ the evolutionary model have some or all of the following characteristics. [5]

- Software intensive systems.
- Have rapidly changing software technology.
- Humans are an integral part of the system.
- Have a large number of diverse users.
- Developing an unprecedented system.
- Limited capability is needed quickly.

2.2.1.4 Spiral Model

The spiral model [6] was developed with the goal of reducing risk in the software life cycle. It combines elements of the waterfall, evolutionary, and incremental models, and depending on how it is implemented can strongly resemble any combination of the others. The model's spiral nature can be seen in Figure 2-5, one of several variants. The

project starts at the center and progresses through multiple cycles, each working through the software development activities associated with the four quadrants:

1. Determine objectives, alternatives, constraints.
2. Evaluate alternatives. Identify and resolve risks.
3. Develop the next-level product.
4. Plan the next phase.

Risk management is a key element of the Spiral model and each round of the spiral identifies problems with the highest risk and develops solutions for that set of problems. The process may even resemble a waterfall with additional risk management techniques. Each cycle ends in a review in which stakeholders agree on plans for the next cycle. While a prototype may be produced for IOC, software is usually not developed for release until the last cycle. [7]

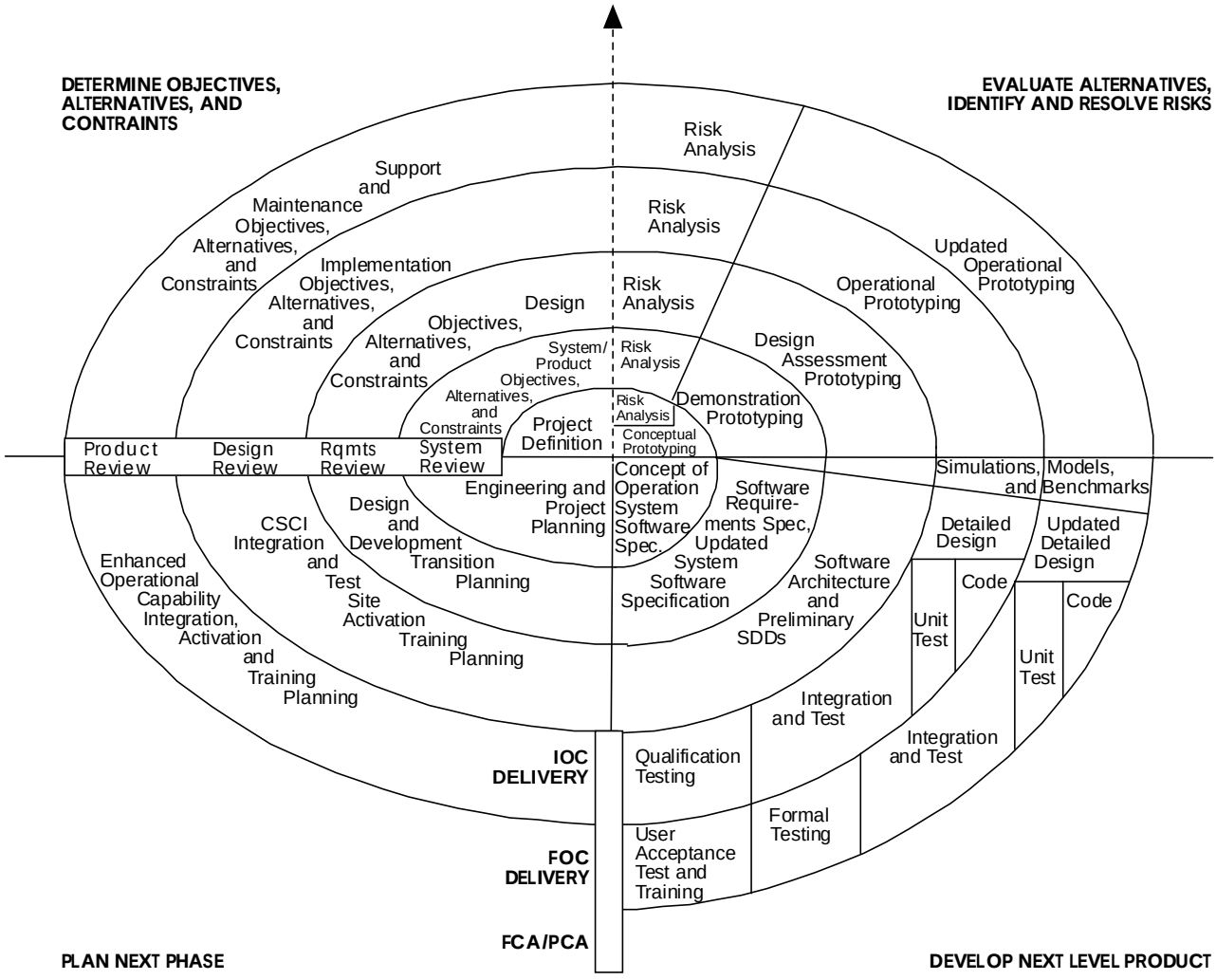

Figure 2-5 The Spiral Model [5]

The Spiral model has been used extensively in the commercial world because of its performance in a market-driven environment. It significantly reduces technical risk and more easily incorporates new technology and innovations. At the same time, it tends to increase cost and schedule risks. In the past the Spiral model has been difficult to implement in the DoD environment. This is because contracts with predefined deliverables and schedules do not easily

accommodate repeating phases, requirement tradeoffs, and changing deliverables. [5] However, use of the Spiral model, where applicable, is directed by DoDI 5000.2, and it has become somewhat popular in the Defense and Aerospace industries.

Advantages

- It provides better risk management than other models.
- Requirements are better defined.
- System is more responsive to user needs.

Disadvantages

- The spiral model is more complex and harder to manage.
- This method usually increases development costs and schedule.

Application

The spiral method should be considered for projects where risks are high, requirements must be refined, and user needs are very important.

2.3 Application

2.3.1 Evolutionary Acquisition and Spiral Development [8]

The Evolutionary Acquisition and Spiral Development approach to acquisition and development has been used to satisfy the preference for evolutionary acquisition strategies established in DoD Directive 5000.1 and DoD Instruction 5000.2. This approach combines incremental, evolutionary, and spiral methods to reduce cycle time and speed delivery of advanced capability to warfighters. It can develop or acquire both hardware and software in manageable pieces and allow the insertion of new technology and capabilities over time.

Evolutionary acquisition fields an initial hardware or software increment (or block) of capability, providing improved capability in a shorter period of time than is possible with a full development effort. This initial deployment is followed by subsequent increments of capability delivered by multiple development cycles. Two variations of this approach are possible. In one, the ultimate capability may be known at the beginning of the program. In the other, the final capability evolves and is defined by matching maturing technologies to the evolving needs of the users.

Spiral development is the iterative process used for developing a defined set of capabilities in a single increment. An increment is a militarily useful and supportable operational capability that can be developed, produced or acquired, deployed, and sustained. An increment may include more than one spiral. Each increment has its own objectives set by the user. An extension of this method can also be applied to Pre-planned Product Improvement (P3I) strategy which adds improved capability to a mature system. This approach is shown in Figure 2-6.

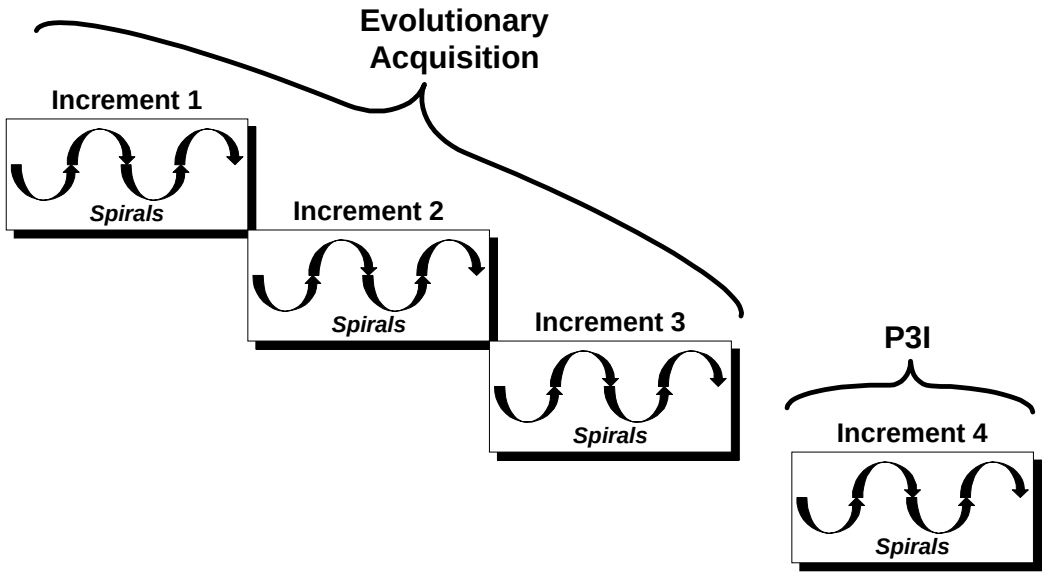

Figure 2-6 Evolutionary Acquisition, Spiral Development, and P3I

2.3.2 Selection Matrix

The purpose of Table 2-1 is to aid in the selection of a life cycle model. It lists various project constraints, along with their applicability to models discussed in this chapter. The applicability is listed for the standard definitions of the models and may not apply equally to modified models. [7] [9]

Table 2-1 Life Cycle Selection Matrix

Legend

Symbol	Meaning
☆	Method is recommended for this constraint
◪	Method is satisfactory for this constraint.
(none)	Method is not recommended for this constraint.

	Waterfall	Incremental	Evolutionary	Spiral
1. Requirements are known and stable.	☆	☆		
2. User needs are unclear/not well defined			☆	◪
3. An early initial operational capability is needed.		☆	☆	
4. Early functionality is needed to refine requirements for subsequent deliveries.		☆	☆	
5. Significant risks need to be addressed.		◪	☆	☆
6. Must interface with other systems.	◪	◪		◪
7. Need to integrate new or future technology.			☆	☆
8. Software is large or complex	☆	☆	◪	◪
9. Software is small or limited in functionality.		◪	◪	◪
10. Software is highly interactive with user.		◪	◪	◪

	Waterfall	Incremental	Evolutionary	Spiral
11. Software involves client/server function.	☑	☑	☑	☆
12. Initial cost and schedule estimates must be followed.	☆	☆		☑
13. Detailed documentation necessary.	☆	☆		☑
14. Minimize impact on current operations.	☆			☑
15. Full system must be implemented.	☆	☑		☑
16. Reduce the number of people required.		☆	☆	
17. Project management must be simpler.	☆	☑		
18. System must be responsive to user needs.		☑	☆	☆
19. Progress must be demonstrated early.		☆	☆	
20. User feedback is needed.		☆	☆	☑
21. Reduce the costs of fixes and corrections.		☑	☆	☑

2.4 Life Cycle Checklist

This checklist is provided to assist you in choosing an appropriate life cycle for your project if you are beginning a development effort, or to ensure you understand your development life cycle if your project is already under way. If you cannot check an item off as affirmative, you need to rectify the situation yourself or get help in that area. [8]

2.4.1 Beginning a Development Project

- ❑ Do you have an understanding of common life cycle models, along with their strengths, weaknesses, and constraints?
- ❑ Has the operational concept been analyzed to determine what life cycle method would best support the acquisition?
- ❑ Can the requirements be fully defined prior to the beginning of the project? Are they stable?
- ❑ Do you know the timeline for deployment of the new system?
- ❑ Are funds secure for development of the entire system?
- ❑ Are the risks identified?
- ❑ Will risks impact the ability of the project to move forward at crucial points in the system development?
- ❑ Is new or developing technology to be used in the system?
- ❑ Will there be a parallel hardware development effort?
- ❑ Do you understand the level of complexity of the system to be developed?
- ❑ Do you know what the interfaces to existing and future systems are?
- ❑ Do you know the size and magnitude of the development effort?
- ❑ Do you understand the users' needs?
- ❑ Are users able or expected to participate in the development?
- ❑ Do you know what types of acquisition contracts are available for this effort?
- ❑ When choosing a life cycle model, do you know why you are choosing it over other models?

2.4.2 Development Project is Under Way

- ❑ Do you know what life cycle model was selected for your project?
- ❑ Do you understand the project aspects pertaining to the life cycle:
  - ❑ Phases – What are they and what are they supposed to accomplish?
  - ❑ Milestones – What are they? What is their significance?
  - ❑ Criteria for transitioning from one phase to another?
  - ❑ Deliverables – What is expected, during phases, and at the end of the project?
  - ❑ Reviews – What is reviewed when? Who are the reviewers? What actions follow a successful review, an unsuccessful review? What are the entry and exit criteria for each review?
  - ❑ Feedback mechanisms – How is feedback obtained? Who provides it? Who receives it? How is it used?
  - ❑ Documentation – What is to be produced and what it is used for?
- ❑ Do you know where your project is in the life cycle?
- ❑ Is your project following the life cycle?

2.5 Regulations

- Clinger-Cohen Act of 1996, The National Defense Authorization Act for Fiscal Year 1996.
- DoD 5000.2-R, *Mandatory Procedures for Major Defense Acquisition Programs and Major Automated Information System Acquisition Programs*, June 2001.
- FAR -- Part 39; *Acquisition of Information Technology*; (FAC 97-27); 25 June 2001.

2.6 References

[1] Brundick, Bill, Editor, *Parametric Cost Estimating Handbook*, Chapter 5, Naval Sea Systems Command, Fall 1995.

[2] Sorensen, Reed, "A Comparison of Software Development Methodologies," *Crosstalk*, January 1995

[3] Whitgift, David, Methods and Tools for Software Configuration Management, 1991, p17.

[4] Pressman, Roger S., "Understanding Software Engineering Practices, Required at SEI Level 2 Process Maturity," Software Engineering Training Series, Software Engineering Process Group, 30 July 1993.

[5] *Guidelines for the Successful Acquisition and Management of Software Intensive Systems* (GSAM), Version 3, Chapter 5, USAF Software Technology Support Center, May 2000.

[6] Boehm, Barry W., "A Spiral Model of Software Development and Enhancement," IEEE Computer, May 1988.

[7] McKenzie, Charlotte A., MIS327 - Systems Analysis and Design, Course Schedule, 1999.

[8] Memorandum from the Undersecretary of Defense for Acquisition Technology and Logistics, Subject: Evolutionary Acquisition and Spiral Development, 12 April 2002.

[9] Quann, Eileen, personal communication to Lloyd K. Mosemann, II, September 1995.

2.7 Resources

2.7.1 Software Life Cycle

Association for Computing Machinery (ACM), ISO 12207 and Related Software Life-Cycle Standards: www.acm.org/tsc/lifecycle.html

Crosstalk magazine articles:

- "Comparison of Software Development Methodologies" www.stsc.hill.af.mil/crosstalk/1995/jan/comparis.asp

- "Spiral Model as a Tool for Evolutionary Acquisition" www.stsc.hill.af.mil/crosstalk/2001/may/boehm.asp
- "Prototypes: Tools That Can Be Used and Misused" www.stsc.hill.af.mil/crosstalk/1994/jan/xt94d01g.asp
- "Extreme Methodologies for an Extreme World" www.stsc.hill.af.mil/crosstalk/2001/jun/leishman.asp
- "Dispelling the Process Myth: Having a Process Does Not Mean Sacrificing Agility or Creativity" www.stsc.hill.af.mil/crosstalk/2001/nov/glazer.asp
- "Balancing Discipline and Flexibility With the Spiral Model and MBASE" www.stsc.hill.af.mil/crosstalk/2001/dec/boehm.asp
- "Customizing the Software Process to Support Avionics Systems Enhancements" www.stsc.hill.af.mil/crosstalk/2001/sep/donzelli.asp
- "Tailoring a Software Process for Software Project Plans Part 1: Context and Making Tailoring Decisions" www.stsc.hill.af.mil/crosstalk/1996/apr/tailorin.asp
- "Tailoring a Software Process for Software Project Plans Part 2: Documenting the Project's Defined Software Process" www.stsc.hill.af.mil/crosstalk/1996/may/tailorin.asp
- "MIL-STD-498: What's New and Some Real Lessons Learned" www.stsc.hill.af.mil/crosstalk/1996/mar/lesslear.asp

DeGrace, Peter and Stahl, Leslie, *Wicked Problems, Righteous Solutions: A Catalogue of Modern Software engineering Paradigms*, Yourdon Press.

Department of Energy (DOE) *Software Engineering Methodology*, Chapter 2: http://cio.doe.gov/sqse/sem_toc.htm

Department of Justice Systems Development Life Cycle Guidance Document: www.usdoj.gov/jmd/irm/lifecycle/table.htm

Georgia Tech slides of software life cycles: www.cc.gatech.edu/computing/SW_Eng/people/Faculty/Colin.Potts/Courses/3302/1-08-mgt/

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, Chapter 5, OO-ALC/TISE, May 2000. Available for download at: www.stsc.hill.af.mil/gsam/guid.asp

LevelA Software, Introduction to life cycle models: www.levela.com/software_life_cycles_swdoc.htm

McKenzie, Charlotte A., MIS327 - Systems Analysis and Design, Course Schedule, 1999. See lecture with model selection guide: http://metr.cua.edu/faculty/mckenzie/mis327/MIS%20327%20Course%20Schedule.html

OSD Program Manager's Guide for Managing Software, Chapter 4 – Life Cycle Models: http://www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Parametric Estimating Handbook, Chapter 6, International Society of Parametric Analysts (ISPA) – Commercial Software Models: www.ispa-cost.org/PEIWeb/ch6.htm

Parametric Cost Estimating Handbook, Chapter 5, DoD, NASA, etc.: www.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm

Vision 7, overview of life cycle phases: http://www.vision7.com/SoftwareDevelopment/ProjectLifeCycle/default.asp

2.7.2 Software Acquisition/Development

Air Force Acquisition Gameboard. Must have a .mil web presence: https://afkm.wpafb.af.mil/ASPs/Gameboard

Air Force Publication Web Site – The official source site for Air Force Administration Publications and Forms: http://afpubs.hq.af.mil

Air Force Software Technology Support Center (STSC) documentation and standards: http://stsc.hill.af.mil/doc/index.asp

Best Manufacturing Practices (BMP), TRIMS Risk Management and Best Practices software downloads: www.bmpcoe.org/pmws/index.html Download KnowHow software and copies of DOD 5000.1, 5000.2, etc: www.bmpcoe.org/pmws/download/knowhow.html

Crosstalk magazine. Extensive database of articles on software engineering. Published by the Air force Software Technology Support Center (STSC): http://www.stsc.hill.af.mil/crosstalk/

Defense Acquisition Deskbook. www.deskbook.osd.mil

Defense Acquisition University web site: www.dau.mil

Department of Defense single stock point of military specifications, standards, and related publications: http://dodssp.daps.mil/

DoD Software Information Clearinghouse: www.dacs.dtic.mil

OSD acquisition web site: www.acq.osd.mil

RSPA. R. S. Pressman & Associates library of software engineering knowledge. In particular, see "An Adaptable Process Model": http://www.rspa.com/spi

Software Program Managers Network (SPMN). Sponsored by the Deputy Under Secretary of Defense for Science and Technology (DUSD (S&T), Software Intensive Systems Directorate. www.spmn.com

SPMN Guidebooks available for download at: www.spmn.com/products_guidebooks.html

University of Southern California (USC) Center for Software Engineering (CSE), Technical Reports: http://sunset.usc.edu/publications/TECHRPTS/index.html

Chapter 3

Project Planning

CONTENTS

This page intentionally left blank.

Chapter 3

Project Planning

3.1 Introduction

Great things do not simply happen. They must be brought to pass through effort. They must also achieve their initial existence in the minds of visionaries before they achieve physical reality. Projects do not often succeed without careful planning. Planning allows the project team to address the five critical factors that determine software program success or failure [1]:

- Quality
- Cost
- Schedule
- Performance
- Supportability

The purpose of planning is to take the project charter, along with the project objectives, and develop a multifaceted plan providing a framework and direction for accomplishing the project, including tasks, budget, schedule, change process, communications, and other various aspects of the project, shown in Figure 3-1. The project plan establishes common goals, expectations, terminology among the participants and stakeholders. To be successful, everyone must work on the same project – following the same plan.

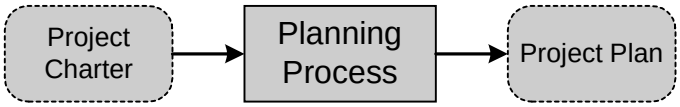

Figure 3-1 Project Planning

Planning is essential to controlling a project. Planning establishes a baseline with which to gauge progress. If the project begins to deviate from the plan, action can be taken to put the project back on track. Without planning, there is no control. [2]

An important purpose of the project plan is to have the project team go through the planning process. This allows the team to work together and rehearse the development of the project. They will evaluate risks, development life cycles, resource availability, budgets, and all significant tasks. They will encounter and solve problems, and get a feel for the work before actually beginning the development work itself. Planning is not the same as having a plan. It consists of an ongoing process by which changes are incorporated in to the project in an orderly manner.

3.2 Process Description

The planning process should be consistent with the life cycle appropriate to the project being performed. In other words, one size doesn't fit all. The activities of the planning process and the contents of the project plan will need to be tailored to the project. Figure 3-2 depicts a generic planning process. Some activities incorporate the output of other activities and must follow them in a sequence. Still other activities, including risk analysis and some not shown, can be performed in parallel.

A well-prepared project plan should minimize change. However, the planning process should extend in some degree beyond the Planning phase to allow necessary changes to be incorporated into the plan. Especially during the planning phase, feedback should exist between different activities to update earlier portions of the project plan according to new understanding gained with the later planning activities.

The activities of the planning process correspond to and produce various portions of the project plan.

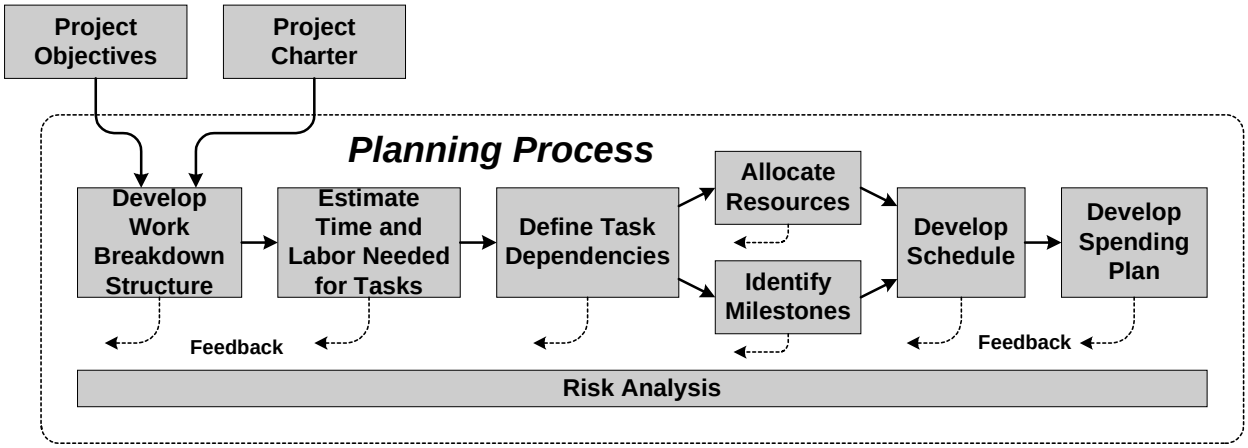

Figure 3-2 Project Planning Process

The project plan is the coordinated compilation of various sub-plans relating to different parts of the project. The following is a list of project plan elements:

- Project Charter
- Work Breakdown Structure
- Task Dependencies
- Network Diagram
- Schedule
- Communications Plan
- Risk Management Plan
- Documentation Plan
- Testing Plan
- Project Objectives
- Labor and Time Estimates
- Resource Allocation
- Assumptions and constraints
- Staffing Plan
- Configuration Management Plan
- Budget/Spending Plan
- Training Plan
- Metrics Analysis Plan

This list is by no means comprehensive. The actual content of a plan, as well as the levels of detail and formality, will depend on the type of project, the deliverables, whether the organization developing the plan is an acquisition or development organization, and requirements to be satisfied by the project. Some of the elements might be considered essential for one type of project and optional for another. Several of these plan elements are described below, while others are covered in greater detail in other chapters.

3.2.1 Project Initialization

3.2.1.1 Project Charter

The project charter should be developed by the tasking organization prior to the formation of a project team (e.g., before the planning phase.) The project charter serves as the starting point for the project plan. It describes why the project has been initiated, what the project will accomplish, how the product or service will be developed or provided, who is responsible to perform the work, and who benefit from the project products or services.

Charter Contents [3]

- Title – Descriptive but not too lengthy or verbose.
- Purpose – Defines the overall scope of the project.
- Statement of Need – Brief statement of the problem the project will solve, but not the solution.

- Expected Results – Description of what will be accomplished when the project is complete. Results must be measurable qualitatively or quantitatively.
- Assumptions – Includes conditions or ground rules which will govern the execution of the project, and which must be acknowledged for its successful completion.
- Roles and Responsibilities – Identify all key players involved with the project, along with their roles and responsibilities. This should include the project manager, the approving authority, and major supporting agencies.
- Authority Statement – Description of the level authority given to the project manager.
- Signature – Signatures of key project stakeholders acknowledging their agreement with the charter.

3.2.1.2 Project Objectives/Requirements

Project objectives and requirements are an essential part of the project plan because they define the overall purpose of the project. In a development effort where all requirements are known at the beginning this part of the plan would be well defined and detailed. In projects where requirements analysis is one of the development phases, the requirements would more likely be top-level, non-detailed objectives. Without some form of objectives there is nothing to plan for.

3.2.2 Project Flow

3.2.2.1 Work Breakdown Structure (WBS)

A work breakdown structure identifies the work to be done to complete the project. It is the key activity of the planning effort. The WBS, per MIL-HDBK-881, should be product-based, including hardware, software, and documentation. DoD programs usually represent the WBS as a hierarchical tree-structure like that shown in Figure 3-3, where level 1 is the overall program, level 2 consists of systems which make up the program, and level 3 is made of subsystems. It may also be represented in an outline form as shown in Figure 3-4. [3]

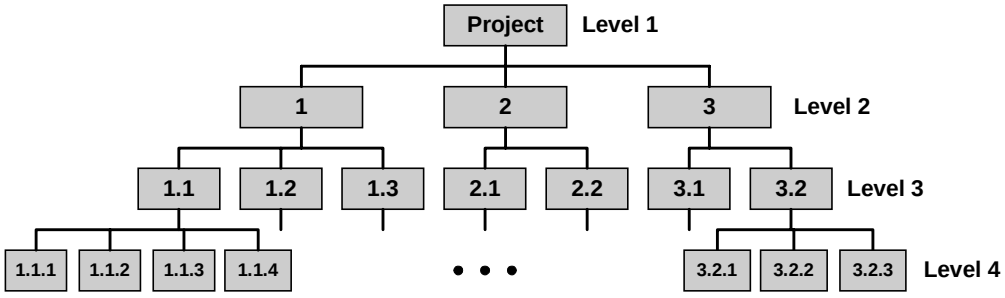

Figure 3-3 Tree Structure WBS

The levels used in the WBS should be appropriate for the project or program. Appendices B and H of MIL-HDBK-881 provide guidance for Electronic/Automated Software Systems and Common Elements respectively. The levels suggested are similar to the levels in this example.

- Level 1 – Total Program
- Level 2 – System
- Level 3 – Subsystem
- Level 4 – Software Build
- Level 5 – Configuration Item

The purpose of the WBS is to describe the whole project and the strategy for completing the project by dividing product into logical components. This makes it much easier to estimate costs, resources, and schedule, as well as assign work and manage the project. [3] Once the sub elements are known for all products, tasks and work activities can be defined to produce the products identified in the

1 - Program

1.1 - System

1.1.1 - Subsystem
1.1.2 - Subsystem
1.1.3 - Subsystem

1.2 - System

1.2.1 - Subsystem
1.2.2 - Subsystem
1.2.3 - Subsystem
1.2.4 - Subsystem

1.3 - System

1.3.1 - Subsystem
1.3.2 - Subsystem
1.3.3 - Subsystem

• • •

Figure 3-4 Outline WBS

WBS.

A well-developed WBS focuses attention on project objectives and allows the project team members to see the whole picture and where their responsibilities fit in. It also facilitates the following benefits: [3]

- Identifying tasks separately from the performing organizations.
- Identifying risks.
- Assigning responsibilities.
- Monitoring time, cost, and performance.
- Developing a network diagram and identifying the critical path or chain of activities.

Application

WBS development follows these steps: [3]

1. Divide the project into manageable pieces.
   - Subdivide major project deliverables into smaller components until they clearly indicate project activities.
   - Rule of thumb: A task need not be smaller than that which can be accomplished in one reporting cycle, usually a week.
   - The product decomposition and associated work activities should reflect the product development lifecycle and processes.
2. Identify the component/task dependencies.
3. Draw it as an outline, hierarchical chart format, or even yellow sticky notes.
4. Validate the work breakdown with key team members and stakeholders.
5. Document each element of the breakdown with its source.
6. Use the WBS, updating it as necessary with the change control process.

3.2.2.2 Network Diagram

The network diagram is usually produced in the form of a chart showing the project activities with their interdependencies. It shows the various activities as chains of events, indicating activity duration and what activities must precede others (see Figure 3-5.) The key feature of the network diagram is the critical path, which identifies the longest duration chain of activities on the project. This determines the minimum time in which the project can be completed. By carefully monitoring and managing critical path activities (an approach known as the *critical path method*) the project manager can be aware of, and perhaps avert, impact to the project schedule. Network diagrams are discussed in greater detail in Chapter 7, Time and Schedule Management.

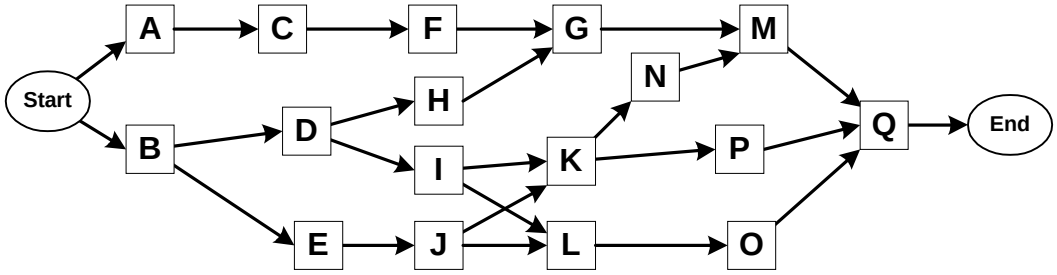

Figure 3-5 Example Network Diagram

3.2.2.3 Schedule

The project schedule describes when project activities start, how long they last, and when they should be finished. Key to developing an accurate schedule is having well defined work packages and task dependencies. A work package is a detailed task description, along with estimates of labor, materials, and time duration needed to perform it.

The task dependencies describe which tasks must precede others. These two inputs allow the scheduler to accurately develop a schedule which shows what should be happening at all times. The schedule should also indicate the critical path, or chain of events which determine the overall duration of the project. The schedule is usually in the form of Gantt charts and milestone charts. [4]

Scheduling is discussed in Chapter 7, Time and Schedule Management.

3.2.2.4 Communications Plan

Another element of the project plan is the communications plan. It identifies the different groups or individuals that must pass information between and among each other. It also defines the type of information being communicated and frequency. This should be documented for the higher levels of reporting, but should only be defined for the working levels as needed. Communications needs will change throughout the project and the plan will need to be modified.

3.2.2.5 Risk Management Plan

Knowing what risks the project faces and managing those risks is essential to successful completion of the project. While some problems come up unexpectedly, most risks can be foreseen and prepared for. A description of possible risks and a plan to avoid or minimize those risks is an essential part of the overall program plan.

Risk analysis and mitigation planning are covered in Chapter 5, Risk Management.

3.2.3 Project Allocations

3.2.3.1 Spending Plan/Budget

Costs information is needed to determine whether the project can be accomplished within the available budget. The important items to include in the spending plan are the costs for accomplishing the different tasks, the total cost of the project, and rate of spending for the different tasks and the whole project. This information will provide a baseline for the project manager to monitor and manage project costs.

Estimating costs and managing budgets are discussed in Chapter 6, Cost Management.

3.2.3.2 Resource Allocation

Projects cannot be performed without resources. The primary resource is usually labor. Other resources may include equipment, materials, transportation, offices, and other facilities such as test, manufacturing, printing, etc. Resource planning should identify what resources are needed in what quantity, and when and how long they are needed.

3.2.3.3 Configuration Management Plan

The configuration management plan defines the process and rules for establishing or baselining the system and for incorporating and documenting changes to the system being developed or to its components.

Configuration management is presented in Chapter 9.

3.2.4 Project Measurement

3.2.4.1 Metrics Analysis Plan

If you can't measure it, you can't manage it. The project measurement plan identifies the aspects of the project to be used as metrics and how those metrics are collected and used.

Measurement and metrics are covered in Chapter 8.

3.2.4.2 Test Plan

The project will include various types of testing throughout its development, integration, and acceptance activity. The test plan defines the testing objectives and strategies for the project. It should describe the testing phases, when testing is to be performed, by whom, and the types of testing to be performed, including unit, interface, integration, regression, system/functional tests, etc.

Testing is covered in chapter 13.

3.2.5 Signoff to Plan

Ideally, the project plan should be developed by representatives of the project team and all major stakeholders. Their input aids in providing a sense of ownership and getting them to support the plan. Where the plan cannot have everyone's input, it is still important to have written agreement from all interested parties to ensure everyone is working to the same plan and to have their support.

3.3 Planning Checklist

This checklist is provided as to assist you in developing a project plan. If you cannot check an item off as affirmative, you need to either rectify the situation or develop a contingency plan to solve problems that may arise.

Before Planning

- ❑ 1. Have you published a list of contacts for all stakeholders and team members?
- ❑ 2. Do you have adequate, unambiguous project objectives or requirements to work with?
- ❑ 3. Do you have an action plan for developing the project plan, including a schedule and a responsibility matrix for participants in the planning activities.
- ❑ 4. Do you know which activities depend on outputs from other activities?
- ❑ 5. Have you documented the planning process?
- ❑ 6. Have you documented all constraints and assumptions? [5]
- ❑ 7. Do you have a written outline listing the contents of your project plan?
- ❑ 8. Have you chosen computer based tools for planning and managing the project?

During Planning

- ❑ 9. Are project developers included in the planning process wherever possible? [5]
- ❑ 10. Are you avoiding *paralysis by analysis*, where things are studied to such a level of detail that action never takes place?
- ❑ 11. Are you following your planning process?
- ❑ 12. Are the defined tasks unambiguous?
- ❑ 13. Does each task have a single point of responsibility?
- ❑ 14. Can each task be performed by an individual or a single team?
- ❑ 15. Is each task associated with a single, continuous time frame?
- ❑ 16. Have you considered holidays, vacations, and training in your schedule and staffing plans? [5]
- ❑ 17. Have you considered project management activities such as planning, meetings, and managing people? [5]
- ❑ 18. Have you considered overhead time such as system down time, outages, or system repairs? [5]

After Planning

- ❑ 19. Is your plan realistic, that is, achievable? [5]
- ❑ 20. Are all stakeholders and participants committed to supporting the project objectives?
- ❑ 21. Have all involved parties formally agreed with the project plan?
- ❑ 22. Does your project scope or any of the objectives need to be modified?
- ❑ 23. Have you documented lessons learned from the planning process?

3.4 Regulations

- MIL-HDBK-881: www.acq.osd.mil/pm/newpolicy/wbs/mil_hdbk_881/mil_hdbk_881.htm

3.5 References

[1] *Guidelines for the Successful Acquisition and Management of Software Intensive Systems* (GSAM), Version 3, Chapter 7, USAF Software Technology Support Center, May 2000.

[2] Lewis, James P., *Fundamentals of Project Management*, Chapter 2, American Management Association, 1997.

[3] Software Technology Support Center Course: *Life Cycle Software Project Management*, Project Initiation, 9 October 2001.

[4] Verzuh, Eric, *The Fast Forward MBA in Project Management*, Chapter 7, John Wiley & Sons, Inc., 1999.

[5] Ambler, Scott W., IBM Developer Library, "Project planning tips", December 2000.

3.6 Resources

Ambler, Scott W., IBM Developer Library, "Project planning tips": www-106.ibm.com/developerworks/library/tip-proj.html

Blair, Gerard M., "Planning A Project,": www.ee.ed.ac.uk/~gerard/Management/art8.html

Crosstalk magazine articles:

- "Strategic Visioning": http://stsc.hill.af.mil/crosstalk/1995/nov/strategi.asp
- "Earned Value Project Management ... an Introduction": http://stsc.hill.af.mil/crosstalk/1999/jul/fleming.asp
- "Tailoring a Software Process for Software Project Plans: Part 1": http://stsc.hill.af.mil/crosstalk/1996/apr/tailorin.asp
- "Applying Management Reserve to Software Project Management": http://stsc.hill.af.mil/crosstalk/1999/mar/lipke.asp
- "DoD Software Acquisition Management Overview": http://stsc.hill.af.mil/crosstalk/1997/apr/dodacquisition.asp
- "Writing an Effective IV&V Plan": http://stsc.hill.af.mil/crosstalk/2000/nov/walters.asp
- "Really Bad Metrics Advice": http://stsc.hill.af.mil/crosstalk/1998/aug/backtalk.asp
- "Tailoring a Software Process for Software Project Plans Part 2: Documenting the Project's Defined Software Process": http://stsc.hill.af.mil/crosstalk/1996/may/tailorin.asp

Department of Energy (DOE) Software Engineering Methodology, Chapter 3: http://cio.doe.gov/sqse/sem_toc.htm

DoD Software Information Clearinghouse, Cost Estimating: www.dacs.dtic.mil/databases/url/key.hts?keycode=4

Department of Justice Systems Development Life Cycle Guidance, Chapters 2 - 5: www.usdoj.gov/jmd/irm/lifecycle/table.htm

DSMC, "Scheduling Guide for Program Managers": www.dsmc.dsm.mil/pubs/gdbks/scheduling_guide.htm

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, Chapter 7, OO-ALC/TISE, May 2000. Available for download at: www.stsc.hill.af.mil/gsam/guid.asp

L. C. Powers web site. Project management resource, Project planning tutorial: www.lcpowers.com/project_planning_tutorial.htm

MAPNP Web Site, Planning in Organizations: www.mapnp.org/library/plan_dec/plan_dec.htm

- Strategic Planning: www.mapnp.org/library/plan_dec/str_plan/str_plan.htm

MindTools, Project Planning & Management Tools: www.mindtools.com/pages/main/newMN_PPM.htm

University of Wollongong. Introduction to PERT & CPM: http://terra.uow.edu.au/buss/buss953/Lectur05.ppt

University of Texas - Project planning tutorial: www.utexas.edu/academic/cit/howto/tutorials/project/planning.html

This page intentionally left blank.

Chapter 4

Requirements Management

CONTENTS

This page intentionally left blank.

Chapter 4

Requirements Management

4.1 Introduction

The goal of any endeavor is defined by its requirements. Requirements define the purpose of a project and are the foundation of the project plan. The stated requirements will determine what is to be produced or achieved. One of the primary reason software projects fail is because requirements are incomplete or incorrect. [1] If the requirements are not done right, the project will probably fail, no matter what is done in subsequent phases. [2] Incomplete or defective requirements result in inaccurate product descriptions and erroneous cost and schedule estimates, which in turn deliver an unusable product. What is not asked for will not be produced, and what is asked for will be produced even if it's not needed. Table 4-1 shows the relative costs of fixing software requirements defects in different phases of the project. Whatever it may cost to do things right in the requirements phase, it will be 3 to 1000 times more costly to fix later if not done right.

Phase in Which Fixed	Relative Cost
Requirements	1
Design	3 – 6
Coding	10
Development Testing	15 – 40
Acceptance Testing	30 – 70
Operations	40 – 1000

Table 4-1 Relative Costs of Fixing Requirement Errors [3]

If requirements are complex and not fully known up front, a life cycle that allows for the refinement of requirements, such as spiral development, should be used. More on this can be found in Chapter 2, Software Life Cycle.

Requirements engineering is a sub-discipline of systems engineering. This chapter presents an overview of requirements and requirements engineering. Because this is such a crucial aspect of any project, it is recommended that additional resources (section 4.6) be used for a greater insight and understanding.

4.2 Process Description

4.2.1 Requirements Definitions

Requirements define capabilities or conditions possessed by a system or a component needed to solve a problem or achieve an objective. They can be divided into two categories, functional and nonfunctional. Functional requirements describe what a system should do in response to a specific stimulus (e.g. when the power switch is first turned to "On" the radio will perform the standard self-check routines.) [4] This is shown in Figure 4-1.

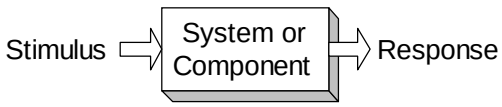

Figure 4-1 Functional Requirements

Nonfunctional requirements include performance and system constraints that affect development and design. They include categories such as these:

- Safety
- Performance
- Survivability
- Reliability
- Environment
- Implementation
- Support
- Security
- People

Requirements must be carefully derived through analysis of user needs and documented. A major point to remember is that requirements should specify *what* is to be done, not *how* it is to be done.

4.2.2 Requirements Process Overview

The requirements engineering process is the primary work of the Requirements phase of the project, but it continues throughout the project to some degree because of requirements changes in later phases. It consists of two major types of activities. The first is requirements development, which is the process of generating requirements from user inputs and analysis. The second is requirements management, which is all other requirements activities which are not part of the development process. Figure 4-2 shows the breakout of the major activities of requirements engineering.

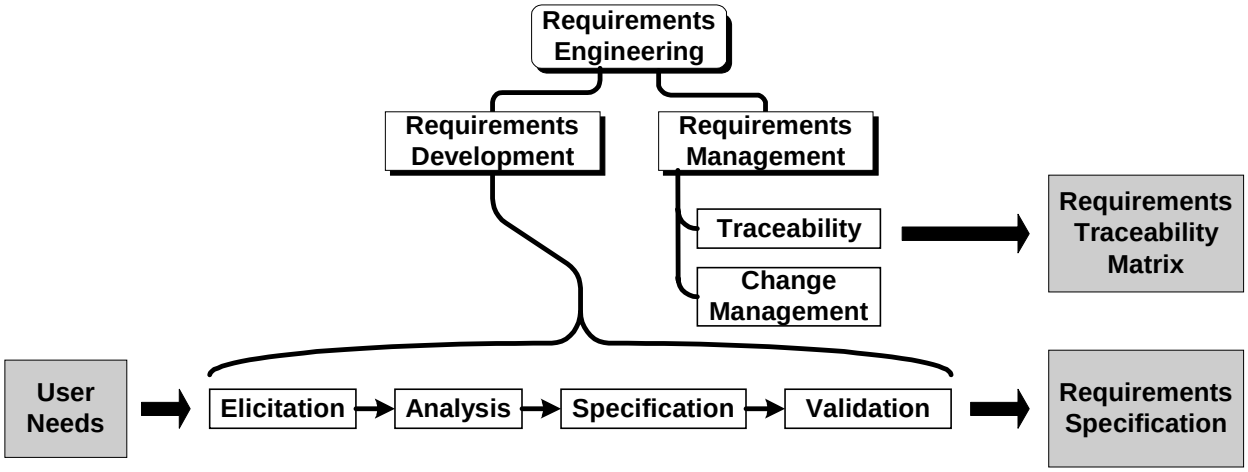

Figure 4-2 Requirements Engineering Process Overview [5]

The process shows user needs being used to generate a Requirements Specification and a Requirements Traceability Matrix (RTM). The four activities of requirements development form a sub-process, while the activities of requirements management do not flow into each other, but are separate. Each of the activities will be discussed greater detail.

4.2.2.1 Elicitation

The elicitation activity consists of gathering information about user needs, primarily from the users themselves. It is a process of helping the users to understand and articulate their requirements so they can make it known to the developers. What is wanted is information about what users are currently working with, why it is inadequate, what their vision of an improved system is, and why. An excellent skill to implement on a grand scale in this area is reflective listening, where you repeat back in your terms what you think you heard, to see if the other party agrees. The general elicitation procedure consists of five steps: [6]

1. Identify relevant requirements sources.
2. Ask them appropriate questions to understand their needs.
3. Look for implications, inconsistencies, and unresolved issues in gathered information.
4. Confirm your understanding of requirements with the users.
5. Synthesize appropriate statements of the requirements.

There are many ways to elicit user requirements, both direct and indirect. The following list presents several methods commonly used. More detail on each of these can be found in the source document, available via the Internet. [7]

- Questionnaire Interviews
- Open-ended Interviews
- Introspection
- Discourse Analysis
- Discussion
- Protocol Analysis
- Brainstorming
- Focus & Application Development Groups
- Surveys

4.2.2.2 Analysis

Requirements analysis is the process of breaking down user requirements into their components and studying these to develop a set of system requirements. The three major goals of this process are: [4]

1. Achieve agreement among developers and customers.
2. Provide a basis for design.
3. Provide a basis for Verification and Validation (V&V)

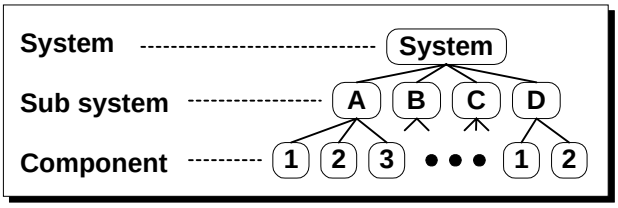

Figure 4-3 Example Requirements Levels

The analysis process consists of developing a set of high-level requirements describing the functionality of the overall system, followed by a stepwise process of decomposing the higher-level requirements into successively more detailed functional and nonfunctional requirements. This decomposition will probably have several levels, as shown in Figure 4-3. In addition to these levels there will probably be different sets of requirements developed for hardware, software, and the interfaces between them. There will probably also be requirements relating to Human-Computer Interaction (HCI).

4.2.2.3 Specification

Specification is the documentation of the requirements developed through requirements analysis. This is a critical activity. Requirements must be carefully worded to state clearly what is wanted. Ambiguous, incomplete, or incorrect requirements can have disastrous results in later phases of development. It is essential to specify requirements in such a way that they are not misinterpreted. They should say what you mean and mean what you say. See Section 4.2.3 for more details on the requirement specification.

4.2.2.4 Validation

Requirements must be carefully validated to ensure that they are complete, correct, and unambiguous. Validation should not be confused with verification, which means to determine whether the product meets the requirements. Validation's purpose is to ensure the requirements describe what the user really needs. It involves the users, the developers, and often an independent validating organization. Once requirements are validated through a formal review process, the requirements specification becomes the baseline for all subsequent development and testing activities. In addition to written documents, requirements are usually maintained in a computer database.

4.2.2.5 Traceability

Traceability is a requirements management activity where requirements are traced back to their original higher-level requirement sources. This ensures that all higher-level requirements are being met by detailed requirements. It also ensures that lower-level requirements have a higher-level source and have not been arbitrarily added to the scope of the project. This effort generally employs some type of database tool and produces the requirements traceability matrix. More on the RTM can be found in Section 4.2.4.

4.2.2.6 Change Management

Requirements will change because of technical issues, funding issues, scheduling issues, and other real-life project events. There must be a process in place to deal with these changes. This process defines the requirements management activity and continues on throughout the remainder of the project. The first responsibility of change management is to determine what changes will be allowed. This is usually accomplished by creating a change board con-

sisting of appropriate developers and stakeholders. All change requests must be reviewed by the board, and only those approved by the board are implemented. The second responsibility of change management is to document the changes and make the appropriate updates to the requirements documents and databases. If requirements are properly traced it facilitates tracking changes through the different levels of specifications.

4.2.3 Requirements Specification

4.2.3.1 Contents

The requirements specification is the primary product of the requirements phase, and the directing document for all development and testing efforts. The system will be built or modified, and tested to comply with this document. Figure 4-4 shows the requirements specification broken out into a top-level system definition of specification, and three major subsystem specifications for hardware, software, and the interface between them. Also shown are those development efforts that are directed by the software requirements specification. Although not shown, there would be similar efforts for the hardware and interfaces.

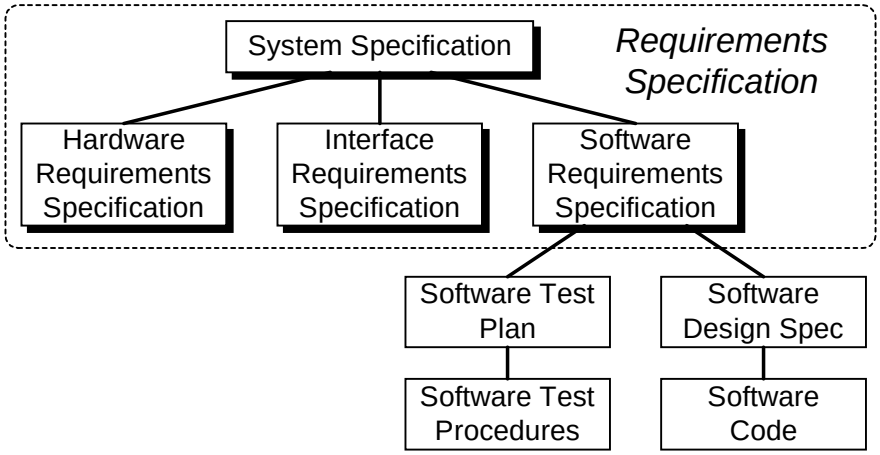

Figure 4-4 Requirements Specification and Follow-on Products

In addition to dividing requirements into different system levels and different disciplines, the requirements specification should contain the following types of requirements and information:

- Functional requirements
  - All inputs to the system.
  - All outputs from the system (data or actions).
  - Information about what must be accomplished between the inputs and outputs (processes within the system.)
  - Method of verification.
- Nonfunctional requirements (reliability, efficiency, maintainability, portability, capacity, etc.) [4]

In software intensive systems these requirements may take the form of natural language, as well as making extensive use of *entity relationship diagrams [8]*, *data flow diagrams [9]*, or other software requirements specification methods.

The requirements specification should not include project requirements, designs, or product assurance plans (Configuration management plan, test plans, validation & verification plans, etc.) [4]

4.2.3.2 Requirements Specification Properties

A good requirements specification should have the following properties: [10]

- Unambiguous
- Consistent
- Usable
- Concise
- Complete
- Modifiable
- Correct
- Feasible
- Verifiable
- Traceable
- Prioritized

4.2.4 Requirements Traceability Matrix

The RTM consists of a database of requirements in a matrix format. Each requirement is numbered and the requirement text is included, along with the source requirement number from higher level documents. When printed out, the RTM becomes a single document with multiple appendices, each appendix containing the trace for a single subsystem specification to a higher specification as shown in Figure 4-5.

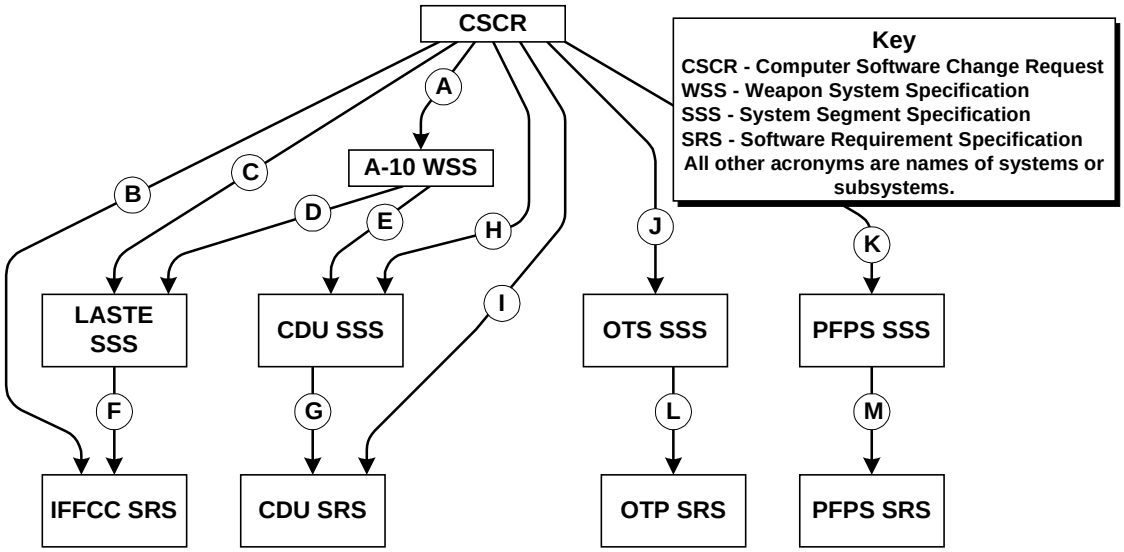

Figure 4-5 Example Requirements Traces Between System Components

In the example shown, the boxes represent different subsystem specifications. The arrows represent the flow down of requirements from higher levels to more detailed levels. The circles contain the letter of the appendix in the RTM which traces the requirements between the two documents.

4.3 Requirements Engineering Checklist

This checklist is provided to assist you in requirements engineering. If you cannot check an item off as affirmative, you need to either rectify the situation or develop a contingency plan to solve problems that may arise.

4.3.1 Requirements Development

- ❑ 1. Have you had extensive user involvement in developing the requirements?
- ❑ 2. Do all stakeholders understand and agree on how the system will be used?
- ❑ 3. Are all stakeholders satisfied with the requirements?
- ❑ 4. Do the developers understand the requirements?
- ❑ 5. Are all requirements clear and unambiguous?
- ❑ 6. Have you distinguished between needs and wants? Are requirements relevant?
- ❑ 7. Are requirements consistent with each other (i.e., they don't conflict.)
- ❑ 8. Are requirements complete? Do the requirements cover everything that is supposed to be accomplished?

- ❑ 9. Has design detail been left out of the requirements?
- ❑ 10. Are all requirements testable?
- ❑ 11. Can you see the requirement as an output?
- ❑ 12. Are all requirements easily recognized as requirements by using the word shall?
- ❑ 13. Have the requirements been prioritized?
- ❑ 14. Are requirements feasible with respect to cost, schedule, and technical capability?
- ❑ 15. Are requirements verifiable?
- ❑ 16. Is the system boundary clearly defined; what is in scope, what is not?
- ❑ 17. Are all external interfaces to the system clearly defined?
- ❑ 18. Is the specification written so that it can be modified when necessary, with minimal impact to the rest of the document?
- ❑ 19. Are you conducting formal and informal reviews of requirements documents?

4.3.2 Requirements Management

- ❑ 20. Have all requirements been entered into the requirements database?
- ❑ 21. Are the requirements traces sorted to allow requirements lookup by paragraph number, requirement number, or other useful index?
- ❑ 22. Can all requirements be traced to original system-level requirements?
- ❑ 23. Are all system-level requirements allocated to lower level, subsystem requirements?
- ❑ 24. Do you have a requirements change process documented and in place?
- ❑ 25. Have you identified members of the requirements change board?
- ❑ 26. Is adequate impact analysis performed for proposed requirements changes?
- ❑ 27. Do you know who is responsible for making the changes?
- ❑ 28. Have requirement changes been traced upward and downward through the higher and lower-level specifications?
- ❑ 29. Do you have a process in place to maintain and control the different versions of the requirements specification?

4.4 Regulations

- AFI 10-601; Mission Needs and Operational Requirements Guidance and Procedures.
- AFPD 10-6; Operational Requirements; Mission Needs and Operational Requirements.
- AFPD 90-11; Planning Systems.
- CJCSI 3170.01A, Requirements Generation System.
- DoD 4120.24-M, Defense Standardization Program (DSP) Policies and Procedures.
- DoD 5000.2-R, Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information System (MAIS) Acquisition Programs, Part 2; 2.2, Requirements.
- DoDD 5000.1, The Defense Acquisition System, 4.2.2. Time-Phased Requirements and Communications with Users.
- DoDI 5000.2, Operation of the Defense Acquisition System, 4.6.1.1. Requirements Generation System and E2.1.18. Requirements Authority.
- FAR 11 Describing Agency Needs and FAR 39 Acquisition of Information Resources.

4.5 References

[1] The Standish Group, "The Scope of Software Development Project Failures," 1995.

[2] DoD, *Program Manager's Guide For Managing Software*, v. 0.6, Chapter 6, 21 June 2001.

[3] Gause, Donald and Weinberg, Gerald, *Exploring Requirements: Quality Before Design*, Dorset House, 1989.

[4] Software Technology Support Center Course: *Life Cycle Software Project Management*, Project Initiation, 9 October 2001.

[5] Wiegers, Karl E., "When Telepathy Won't Do: Requirements Engineering Key Practices," Cutter IT Journal, May 2000. www.processimpact.com/articles/telepathy.html

[6] Raghavan, Zelesnik, & Ford, "Lecture Notes of Requirements Elicitation," 1994: www.sei.cmu.edu/publications/documents/ems/94.em.010.html

[7] Henderson, Lisa G.R., "Requirements Elicitation in Open-Source Programs", *Crosstalk*, 2000, www.stsc.hill.af.mil/crosstalk/2000/jul/henderson.asp

[8] SmartDraw, Entity Relationship Diagram tutorial: www.smartdraw.com/resources/centers/software/erd.htm

[9] SmartDraw, Data Flow Diagram tutorial: www.smartdraw.com/resources/centers/software/dfd.htm

[10] Davis, Alan M., *Software Requirements Analysis and Specification*, Prentice-Hall, 1990.

4.6 Resources

Christel & Kang, "Issues in Requirements Elicitation," 1992: www.sei.cmu.edu/publications/documents/92.reports/92.tr.012.html

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "Writing Effective Natural Language Requirements Specifications": www.stsc.hill.af.mil/crosstalk/1999/feb/wilson.asp
- "Experiences in the Adoption of Requirements Engineering Technologies": www.stsc.hill.af.mil/crosstalk/1998/dec/cook.asp
- "An Examination of the Effects of Requirements Changes on Software Releases": www.stsc.hill.af.mil/crosstalk/1998/dec/stark.asp
- "Doing Requirements Right the First Time": www.stsc.hill.af.mil/CrossTalk/1998/dec/hammer.asp
- "Four Roads to Use Case Discovery": www.stsc.hill.af.mil/CrossTalk/1998/dec/ham.asp
- "An Examination of the Effects of Requirements Changes on Software Releases": www.stsc.hill.af.mil/CrossTalk/1998/dec/stark.asp
- "Experiences in the Adoption of Requirements Engineering Technologies": www.stsc.hill.af.mil/CrossTalk/1998/dec/cook.asp
- "Recommended Requirements Gathering Practices": www.stsc.hill.af.mil/crosstalk/2002/apr/young.asp
- "Making Requirements Management Work for You": www.stsc.hill.af.mil/crosstalk/1999/apr/davis.asp

Department of Energy (DOE) Software Engineering Methodology, Chapter 4: http://cio.doe.gov/sqse/sem_toc.htm

Department of Energy, Software Risk Management Practical Guide: http://cio.doe.gov/sqse/pm_req.htm

Department of Justice Systems Development Life Cycle Guidance, Chapter 6: www.usdoj.gov/jmd/irm/lifecycle/table.htm

IEEE Computer Society: http://computer.org

- 4th International Conference on Requirements Engineering (ICRE'00): http://computer.org/proceedings/icre/0565/0565toc.htm
- 3rd International Conference on Requirements Engineering (ICRE'98): http://computer.org/proceedings/icre/8356/8356toc.htm
- 2nd International Conference on Requirements Engineering (ICRE '96): http://computer.org/proceedings/icre/7252/7252toc.htm

INCOSE Requirements Working Group: www.incose.org/rwg/

- "Factors Influencing Requirement Management Toolset Selection": www.incose.org/lib/rmtools.html

- "Characteristics of Good Requirements": www.incose.org/rwg/goodreqs.html
- "Requirements Management Technology Overview": www.incose.org/tools/reqsmgmt.html
- "What is a Requirement?": www.incose.org/rwg/what_is.html

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, Chapter 11, OO-ALC/TISE, May 2000. Available for download at: www.stsc.hill.af.mil/gsam/guid.asp

Heimdahl and Associates, "Requirements Verification Checklist": http://www-users.cs.umn.edu/~heimdahl/csci5802/memos/RequirementsChecklist.pdf

Hooks, Ivy F., "Why Johnny Can't Write Requirements": www.complianceautomation.com/papers/whyjohnny.htm

Leffingwell, Dean A., "A Field Guide to Effective Requirements Management Under SEI's Capability Maturity Model": www.rational.com/products/whitepapers/299.jsp

Ludwig, J., Requirements management site: www.jiludwig.com

Modell, Martin, *A Professional's Guide to Systems Analysis, 2ed.*, full book online: http://www.dai-sho.com/pgsa2/index.html

Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapter 6: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Rational Software Corp., *UML Notation Guide 1.1*, 1997: http://curie.informatik.fh-luebeck.de/~st/UML/UML1.1/

Requirements Management Place, The: www.rmplace.org

Requirements Generation System, Joint Chiefs of Staff: www.jsc.mil/jsce3/EMCSLSA/stdlib/cd/Added/3170_01b.pdf

Wiegers, Karl E.,

- "10 Requirements Traps to Avoid": www.processimpact.com/articles/reqtraps.html
- "Writing Quality Requirements": www.processimpact.com/articles/qualreqs.html
- "First Things First: Prioritizing Requirements": www.processimpact.com/articles/prioritizing.html
- "Automating Requirements Management": www.processimpact.com/articles/rm_tools.html
- "In Search of Excellent Requirements": http://www.processimpact.com/articles/exc_reqs.html
- "Habits of Effective Analysts": www.processimpact.com/articles/analyst_habits.html

Chapter 5

Risk Management

CONTENTS

This page intentionally left blank.

Chapter 5

Risk Management

5.1 Introduction

Risk is a product of the uncertainty of future events and is a part of all activity. It is a fact of life. We tend to stay away from those things that involve high risk to things we hold dear. When we cannot avoid risk, we look for ways to reduce the risk or the impact of the risk upon our lives. But even with careful planning and preparation, risks cannot be completely eliminated because they cannot all be identified beforehand. Even so, risk is essential to progress. The opportunity to succeed also carries the opportunity to fail. It is necessary to learn to balance the possible negative consequences of risk with the potential benefits of its associated opportunity. [1]

Risk may be defined as the possibility to suffer damage or loss. The possibility is characterized by three factors: [1]

1. The probability or likelihood that loss or damage will occur.
2. The expected time of occurrence.
3. The magnitude of the negative impact that can result from its occurrence.

The seriousness of a risk can be determined by multiplying the probability of the event actually occurring by the potential negative impact to the cost, schedule, or performance of the project:

Risk Severity = Probability of Occurrence • Potential Negative Impact

Thus, risks where probability is high and potential impact is very low, or vice versa, are not considered as serious as risks where both probability and potential impact are medium to high.

Project managers recognize and accept the fact that risk is inherent in any project. They also recognize that there are two ways of dealing with risk. One, risk management, is proactive and carefully analyzes future project events and past projects to identify potential risks. Once risks are identified, they are dealt with by taking measures to reduce their probability or reduce the impact associated with them. The alternative to risk management is crises management. It is a reactive and resource-intensive process, with available options constrained or restricted by events. [1]

Effective risk management requires establishing and following a rigorous process. It involves the entire project team, as well as requiring help from outside experts in critical risk areas (e.g., technology, manufacturing, logistics, etc.) Because risks will be found in all areas of the project and will often be interrelated, risk management should include hardware, software, integration issues, and the human element. [2]

5.2 Process Description

Various paradigms are used by different organizations to organize their risk manage management activities. A commonly used approach is shown in Figure 5-1. While there are variations in the different paradigms, certain characteristics are universally required for the program to be successful. These are listed below: [2]

- The risk management process is planned and structured.
- The risk process is integrated with the acquisition process.
- Developers, users, procurers, and all other stakeholders work together closely to implement the risk process.

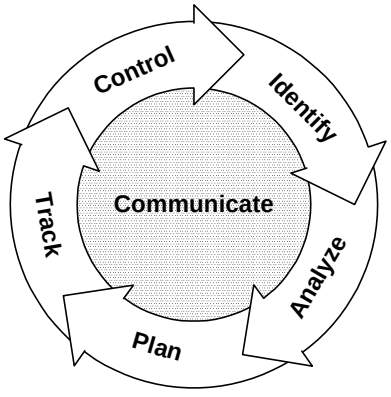

Figure 5-1 Software Engineering Institute Risk Management paradigm [3]

- Risk management is an ongoing process, with continual monitoring and reassessment.
- A set of success criteria is defined for all cost, schedule, and performance elements of the project.
- Metrics are defined and used to monitor effectiveness of risk management strategies.
- An effective test and evaluation program is planned and followed.
- All aspects of the risk management program are formally documented.
- Communication and feedback are an integral part of all risk management activities.

While your risk management approach should be tailored to the needs your project, It should incorporate these fundamental characteristics. The process is iterative and should have all the components shown in Figure 5-2. Note that while planning appears as the first step, there is a feedback loop from the monitoring activity that allows planning and the other activities to be redone or controlled by actual results, providing continual updates to the risk management strategy.

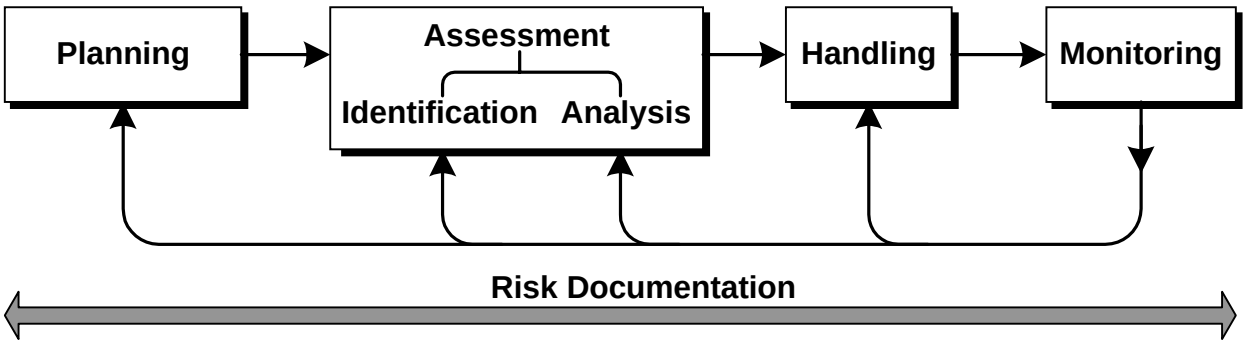

Figure 5-2 - Risk Management Process Example

In essence, the process is a standard approach to problem solving:

1. Plan or define the problem solving process.
2. Define the problem.
3. Work out solutions for those problems.
4. Track the progress and success of the solutions.

5.2.1 Planning

Risk planning includes developing and documenting a structured, proactive, and comprehensive strategy to deal with risk. Key to this activity is the establishment of methods and procedures to do the following:

- Establishing an organization to take part in the risk management process.
- Identify and analyze risks.
- Develop risk-handling plans.
- Monitoring or tracking risk areas.
- Assigning resources to deal with risks.

A generic example risk management plan can be found in Appendix B of the *Risk Management Guide for DoD Acquisition*. (See resources.)

5.2.2 Assessment

Risk assessment involves two primary activities, risk identification and risk analysis. Risk identification is actually begun early in the planning phase and continues throughout the life of the project. The following methods are often used to identify possible risks: [1]

- Brainstorming.
- Evaluations or inputs from project stakeholders.
- Periodic reviews of project data.
- Questionnaires based on taxonomy, the classification of product areas and disciplines.
- Interviews based on taxonomy.
- Analysis of the Work Breakdown Structure (WBS).
- Analysis of historical data.

When identifying a risk it is essential to do so in a clear and concise statement. It should include three components: [1]

1. Condition - A sentence or phrase briefly describing the situation or circumstances that may have caused concern, anxiety, or uncertainty.
2. Consequence – A sentence describing the key negative outcomes that may result from the condition.
3. Context – Additional information about the risk to ensure others can understand its nature, especially after the passage of time.

The following is an example of a risk statement [1]:

Condition	End users submit requirements changes even though we're in the design phase and the requirements have been baselined.
Consequence	Changes could extend system design cycle and reduce available coding time.
Probability & Impact	80%. $2 million.
Mitigation Actions	Who, what, and when?

The other half of assessment is risk analysis. This is the process of examining each risk to refine the risk description, isolate the cause, quantify the probability of occurrence, and determine the nature and impact of possible effects. The result of this process is a list of risks rated and prioritized according to their probability of occurrence, severity of impact, and relationship to other risk areas. [2]

Once risks have been defined, with probability of occurrence and consequences assigned, the risk can be rated as to its severity. This facilitates prioritizing risks and deciding what level of resources to devote to the risk. Figure 5-3 depicts an assessment model using risk probability and consequence levels in a matrix to determine a level of risk severity. In addition to an overall method of risk rating, the model also gives good examples of probability levels, and types and levels of consequences. The ratings given in the assessment guide matrix are suggested minimum ratings. It may be necessary to adjust the moderate and high thresholds to better coincide with the type of project.

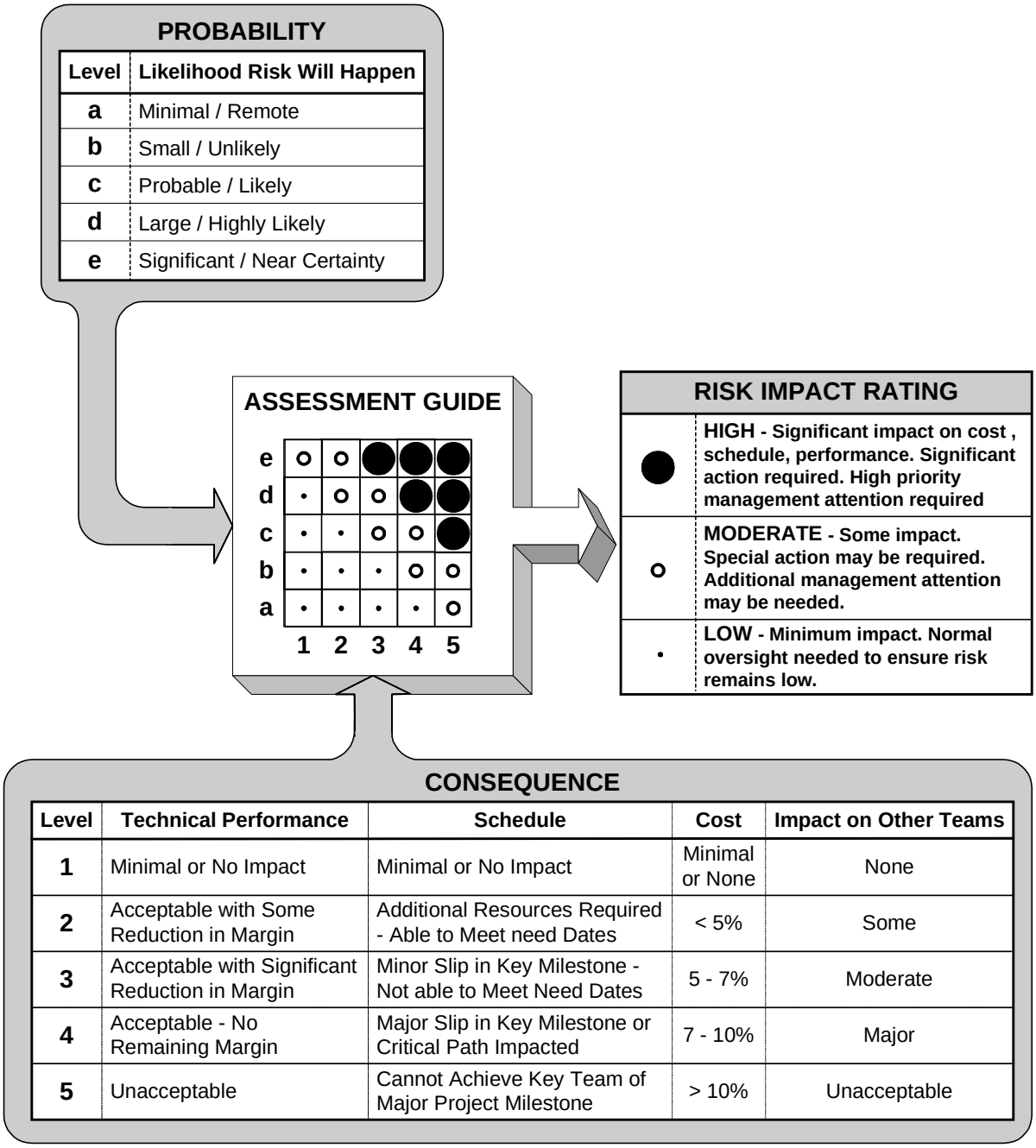

Figure 5-3 Defense Acquisition University Assessment Model [4]

5.2.3 Handling

Risk handling is the process that identifies, evaluates, selects, and implements options for mitigating risks, as shown in Figure 5-3. Two approaches are used in handling risk. The first is to employ options that reduce the risk itself. This usually involves a change in current conditions to lessen the probability of occurrence. The second approach, often employed where risk probability is high, is to use options that reduce the negative impact to the project if the risk condition should occur. Improving jet engine maintenance and inspection procedures to reduce the risk of in-flight engine failure is an example of the first approach. Providing a parachute for the pilot, to reduce loss if the risk condition should occur, is an example of the second.

Figure 5-3 Risk Handling Process

5.2.4 Monitoring

Risk monitoring is the process of continually tracking risks and the effectiveness of risk handling options to ensure risk conditions do not get out of control. This is done by knowing the baseline risk management plans, understanding the risks and risk handling options, establishing meaningful metrics, and evaluating project performance against the established metrics, plans, and expected results throughout the acquisition process. Continual monitoring also enables the identification of new risks that may become apparent over time. It also discovers the interrelationships between various risks. [2]

The monitoring process provides feedback into all other activities to improve the ongoing, iterative risk management process for the current and future projects.

5.2.5 Documentation

Risk documentation is absolutely essential for the current, as well as future, projects. It consists of recording, maintaining, and reporting risk management plans, assessments, and handling information. It also includes recording the results of risk management activities, providing a knowledge base for better risk management in later stages of the project and in other projects. [2] Documentation should include as a minimum the following information:

- Risk management plans.
- Project metrics to be used for risk management.
- Identified risks and their descriptions.
- The probability, severity of impact, and prioritization of all known risks.
- Description of risk handling options selected for implementation.
- Project performance assessment results, including deviations from the baseline plans.
- A record of all changes to the above documentation, including newly identified risks, plan changes, etc.

5.3 Risk Management Checklist

This checklist is provided as to assist you in risk management. If you answer “no” to any of these questions you should examine the situation carefully for the possibility of greater risks to the project. This is only a cursory checklist for such an important subject. Please see the reference documents for more detailed checklists. [5] [6]

- ❑ 1. Do you have a comprehensive, planned, and documented approach to risk management?
- ❑ 2. Are all major areas/disciplines represented on your risk management team?
- ❑ 3. Is the project manager experienced with similar projects?
- ❑ 4. Do the stakeholders support disciplined development methods that incorporate adequate planning, requirements analysis, design, and testing?
- ❑ 5. Is the project manager dedicated to this project, and not dividing his or her time among other efforts?
- ❑ 6. Are you implementing a proven development methodology?
- ❑ 7. Are requirements well defined, understandable, and stable?
- ❑ 8. Do you have an effective requirements change process in place and do you use it?
- ❑ 9. Does your project plan call for tracking/tracing requirements through all phases of the project?
- ❑ 10. Are you implementing proven technology?

- ❑ 11. Are suppliers stable, and do you have multiple sources for hardware and equipment?
- ❑ 12. Are all procurement items needed for your development effort short-lead time items (no long-lead items?)
- ❑ 13. Are all external and internal interfaces for the system well defined?
- ❑ 14. Are all project positions appropriately staffed with qualified, motivated personnel?
- ❑ 15. Are the developers trained and experienced in their respective development disciplines (i.e. systems engineering, software engineering, language, platform, tools, etc.?)
- ❑ 16. Are developers experienced or familiar with the technology and the development environment?
- ❑ 17. Are key personnel stable and likely to remain in their positions throughout the project?
- ❑ 18. Is project funding stable and secure?
- ❑ 19. Are all costs associated with the project known?
- ❑ 20. Are development tools and equipment used for the project state-of-the-art, dependable, and available in sufficient quantity, and are the developers familiar with the development tools?
- ❑ 21. Are the schedule estimates free of unknowns?
- ❑ 22. Is the schedule realistic to support an acceptable level of risk?
- ❑ 23. Is the project free of special environmental constraints or requirements?
- ❑ 24. Is your testing approach feasible and appropriate for the components and system?
- ❑ 25. Have acceptance criteria been established for all requirements and agreed to by all stakeholders?
- ❑ 26. Will there be sufficient equipment to do adequate integration and testing?
- ❑ 27. Has sufficient time been scheduled for system integration and testing?
- ❑ 28. Can software be tested without complex testing or special test equipment?
- ❑ 29. Is the system being developed by a single group in one location?
- ❑ 30. Are subcontractors reliable and proven?
- ❑ 31. Is all project work being done by groups over which you have control?
- ❑ 32. Are development and support teams all collocated at one site?
- ❑ 33. Is the project team accustomed to working on an effort of this size (neither bigger nor smaller?)

5.4 Regulations

- DCMA Directive 1, Contract Management "One Book", 0.0 -- Operating Principles.
- DoD 5000.2-R, Mandatory Procedures for Major Defense Acquisition Programs (MDAPs) and Major Automated Information System (MAIS) Acquisition Programs, Part 2; 2.5, Risk, and Part 6; 6.7, Technology Protection.
- DoD Directive 5000.1 The Defense Acquisition System (October 23, 2000).
- DoD Manual 5000.4-M, Cost Analysis Guidance and Procedures.
- DoDD 5000.4, OSD Cost Analysis Improvement Group (CAIG).
- FAR -- Part 39; Acquisition of Information Technology; (FAC 97-27); 25 June 2001.
- OMB Circular A-109, Major System Acquisition, Para 7, Major System Acquisition Management Objectives.
- Supplement to OMB Circular A-11, Part 3, Planning, Budgeting, and Acquisition of Capital Assets, Appendix Six, Risk Management in the Procurement Phase.
- The Information Technology Acquisition Reform act of 1996 (also known as the Clinger-Cohen Act).

5.5 References

[1] Software Technology Support Center Course: *Life Cycle Software Project Management*, Project Initiation, 9 October 2001.

[2] *Risk Management Guide for DoD Acquisition*, Chapter 2, February 2001. www.dsmc.dsm.mil/pubs/gdbks/risk_management.htm

[3] Higuera, Ron & Haimes, Yacov, "Software Risk Management," Technical Report, 1996. www.sei.cmu.edu/publications/documents/96.reports/96.tr.012.html

[4] *Risk Management Guide for DoD Acquisition*, Appendix B, February 2001.

[5] Arizona State University (ASU), "Question List for Software Risk Identification." www.eas.asu.edu/~riskmgmt/qlist.html

[6] Department of Energy, Risk Assessment Questionnaire. http://cio.doe.gov/sqse/pm_risk.htm

5.6 Resources

Arizona State University software risk management resources: www.eas.asu.edu/~riskmgmt/

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, Chapter 6, OO-ALC/TISE, May 2000. Download at: www.stsc.hill.af.mil/gsam/guid.asp

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "A Practical Approach to Quantifying Risk Evaluation Results": www.stsc.hill.af.mil/crosstalk/2000/feb/hantos.asp
- "A Risk Management Bibliography": www.stsc.hill.af.mil/crosstalk/1994/mar/xt94d03k.asp
- "Continuing Risk Management at NASA": www.stsc.hill.af.mil/crosstalk/2000/feb/rosenberg.asp
- "Identifying and Managing Risks for Software Process Improvement": www.stsc.hill.af.mil/crosstalk/2000/feb/risk.asp
- "Managing Risk Management": www.stsc.hill.af.mil/crosstalk/1999/jul/neitzel.asp
- "Managing Risk with TSP": www.stsc.hill.af.mil/crosstalk/2000/jun/webb.asp
- "Risk Management in Practice": www.stsc.hill.af.mil/crosstalk/1997/apr/management.asp
- "Team Risk Management": www.stsc.hill.af.mil/crosstalk/1995/jan/teamrisk.asp

Department of Energy, Software Risk Management Practical Guide: http://cio.doe.gov/sqse/pm_risk.htm

Department of Justice, *Systems Development Life Cycle Guidance Document*, Risk Management, Chapter 3: www.usdoj.gov/jmd/irm/lifecycle/table.htm

Higuera, Dorofee, Walker, & Williams, "Team Risk Management: A New Model for Customer Supplier Relationships," 1994. Download at: www.sei.cmu.edu/publications/documents/94.reports/94.sr.005.html

Higuera, Ron, et. Al., *Continuous Risk Management Guidebook*, 1996, CMU.

Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapter 5: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Risk Management Guide for DoD Acquisition, 2001. Download at: www.dsmc.dsm.mil/pubs/gdbks/risk_management.htm

Risk Radar, Software for managing risk. Available at: www.iceincusa.com/rskrdr.htm

Software Engineering Institute, Risk management overview: www.sei.cmu.edu/programs/sepm/risk/

Software Engineering Institute , risk management frequently asked questions: www.sei.cmu.edu/programs/sepm/risk/risk.faq.html

US Treasury, Systems Development Life Cycle Handbook, Risk Management Processes, Chapter 5. Download: www.customs.ustreas.gov/contract/modern/sdlcpdfs/tocsdlc.htm

This page intentionally left blank.

Chapter 6

Cost Management

CONTENTS

This page intentionally left blank.

Chapter 6

Cost Management

6.1 Introduction

Cost is one of the three pillars supporting project success or failure, the other two being schedule and performance. Projects that go significantly "over budget" are often terminated without achieving the project goals because stakeholders simply run out of money or perceive additional expenditures as "throwing good money after bad." Projects that stay within budget are the exception, not the rule. A project manager who can control costs while achieving performance and schedule goals should be viewed as somewhat of a hero, especially when we consider that cost, performance, and schedule are closely interrelated.

The level of effort and expertise needed to perform good cost management are seldom appreciated. Too often, there is the pressure to come up with estimates within too short a period of time. When this happens, there is not enough time to gather adequate historical data, select appropriate estimating methods, consider alternatives, or carefully apply proper methods. The result is estimates that lean heavily toward guesswork. The problem is exacerbated by the fact that estimates are often not viewed as estimates but more as actual measurements made by some time traveler from the future. Estimates, once stated, have a tendency to be considered facts. Project managers must remember that estimates are the best guesses by estimators under various forms of pressure and with personal biases. They must also be aware of how others perceive these estimates.

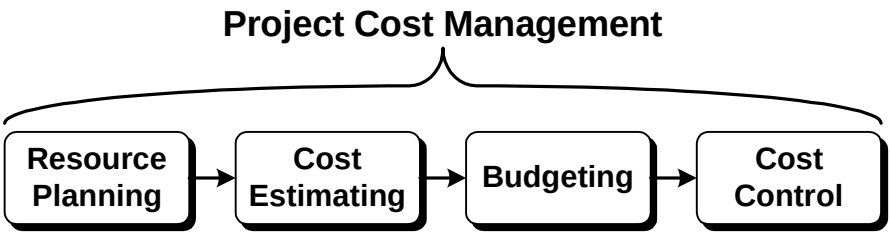

Figure 6-1 Cost Management Processes

Cost management consists of the four main activities or processes shown in Figure 6-1. It requires an understanding of costs far beyond the concepts of money and numbers. Cost of itself can be only measured, not controlled. Costs are one-dimensional representations of three-dimensional objects traveling through a fourth dimension, time. The real-world things that cost represents are people, materials, equipment, facilities, transportation, etc. Cost is used to monitor performance or use of real things but it must be remembered that management of those real things determines cost, and not vice versa.

6.2 Process Description

The first three cost management processes are completed, with the exception of updates, during the project planning phase. The final process, controlling costs, is ongoing throughout the remainder of the project. Each of these processes is summarized below.

6.2.1 Resource Planning

Cost management is begun by planning the resources that will be used to execute the project. Figure 6-2 shows the inputs, tools, and product of this process. All the tasks needed to achieve the project goals are identified by analyzing the deliverables described in the Work Breakdown Structure (WBS). The planners use this along with historical information from previous similar projects, available resources, and activity duration estimates to develop resource requirements. It is important to get experienced people involved with this activity, as noted by the "expert judgment" listed under Tools. They will know what works and what doesn't work.

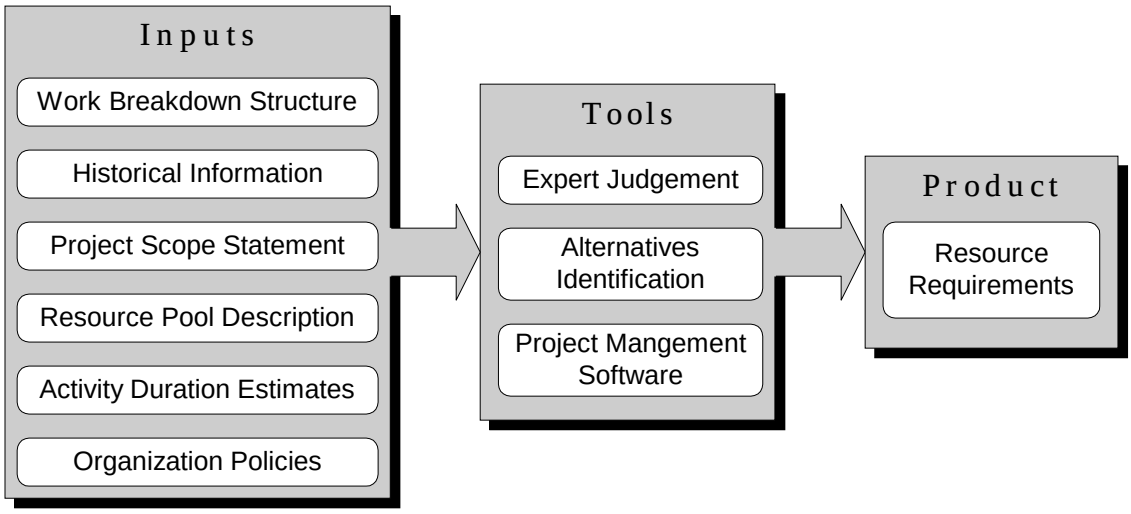

Figure 6-2 Resource Planning Elements [1]

In trying to match up resources with tasks and keep costs in line, the planners will need to look at alternatives in timing and choosing resources. They will need to refer back to project scope and organizational policies to ensure plans meet with these two guidelines.

Except for very small projects, trying to plan without good project management software is tedious and subject to errors, both in forgetting to cover all tasks and in resource and cost calculations.

The output of this process is a description of the resources needed, when they are needed, and for how long. This will include all types of resources, people, facilities, equipment, and materials. Once there is a resource plan, the process of estimating begins.

6.2.2 Estimating Costs

Estimating is the process of determining the expected costs of the project. It is a broad science with many branches and several popular, and sometimes disparate, methods. There are overall strategies to determining the cost of the overall project, as well as individual methods of estimating costs of specific types of activity. Several of these can be found in the resources listed at the end of the chapter. In most software development projects the majority of the cost pertains to staffing. In this case, knowledge of the pay rates (including overhead) of the people working on the project, and being able to accurately estimate the number of people needed and the time necessary to complete their work will produce a fairly accurate project cost estimate. Unfortunately, this is not as simple as it sounds. Most project estimates are derived by summing the estimates for individual project elements. Several general approaches to estimating costs for project elements are presented here. [3] Your choice of approach will depend on the time, resources, and historical project data available to you. The cost estimating process elements are shown in Figure 6-3.

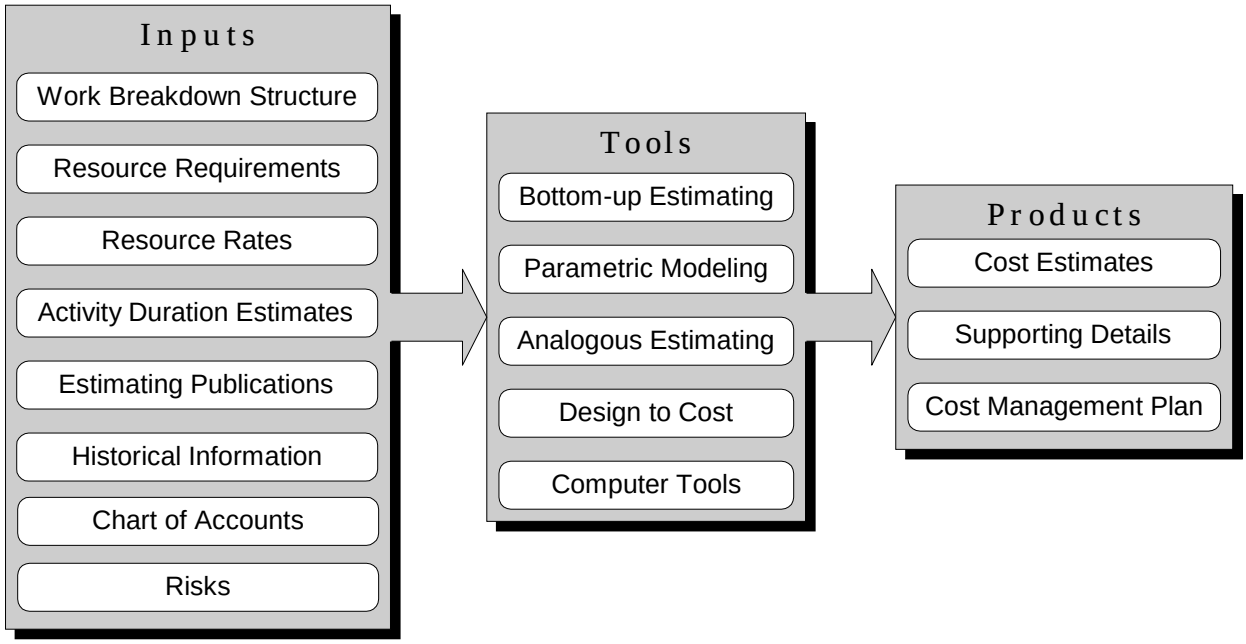

Figure 6-3 Cost Estimating Elements [1]

Cost estimating uses the resource requirements, resource cost rates, and the activity duration estimates to calculate cost estimates for each activity. Estimating publications, historical information, and risk information are used to help determine which strategies and methods would yield the most accurate estimates. A chart of accounts may be needed to assign costs to different accounting categories. A final, but very important, input to the estimating process is the WBS. Carefully comparing activity estimates to the activities listed in the WBS will serve as a reality check and discover tasks that may have been overlooked or forgotten.

The tools used to perform the actual estimating can be one or more of several types. The major estimating approaches shown in Figure 6-3 are discussed here. While other approaches are used, they can usually be classed as variations of these. One caution that applies to all estimating approaches: If the assumptions used in developing the estimates are not correct, any conclusions based on the assumptions will not be correct either.

6.2.2.1 Bottom-Up Estimating

Bottom-up estimating consists of examining each individual work package or activity and estimating its costs for labor, materials, facilities, equipment, etc. This method is usually time consuming and laborious but usually results in accurate estimates if well prepared, detailed input documents are used. [3]

6.2.2.2 Analogous Estimating

Analogous estimating, also known as top-down estimating, uses historical cost data from a similar project or activities to estimate the overall project cost. It is often used where information about the project is limited, especially in the early phases. Analogous estimating is less costly than other methods but it requires expert judgment and true similarity between the current and previous projects to obtain acceptable accuracy. [1]

6.2.2.3 Parametric Estimating

Parametric estimating uses mathematical models, rules of thumb, or Cost Estimating Relationships (CERs) to estimate project element costs. CERs are relationships between cost and measurements of work, such as the cost per line of code. [3] Parametric estimating is usually faster and easier to perform than bottom-up methods but it is only accurate if the correct model or CER is used in the appropriate manner.

6.2.2.4 Design-to-Cost Estimating

Design-to-cost methods are based on cost unit goals as an input to the estimating process. Tradeoffs are made in performance and other systems design parameters to achieve lower overall system costs. A variation of this method

is **cost-as-the-independent-variable**, where the estimators start with a fixed system-level budget and work backwards, prioritizing and selecting requirements to bring the project scope within budget constraints. [3]

6.2.2.5 Computer Tools

Computer tools are used extensively to assist in cost estimation. These range from spreadsheets and project management software to specialized simulation and estimating tools. Computer tools reduce the incidence of calculation errors, speed up the estimation process, and allow consideration of multiple costing alternatives. [1] One of the more widely used computer tools for estimating software development costs is the Constructive Cost Model (COCOMO). The software and users manual are available for download without cost (see COCOMO in the Resources.) However, please note that most computer tools for developing estimates for software development use either lines of code or function points as input data. If the number of lines of code or function points cannot be accurately estimated, the output of the tools will not be accurate. The best use of tools is to derive ranges of estimates and gain understanding of the sensitivities of those ranges to changes in various input parameters.

The outputs of the estimating process include the project cost estimates, along with the details used to derive those estimates. The details usually define the tasks by references to the WBS. They also include a description of how the cost was derived, any assumptions made, and a range for estimate (e.g. $20,000 +/- $2000.) Another output of the estimating process is the Cost Management Plan. This plan describes how cost variances will be managed, and may be formal or informal. [1] The following information may be considered for inclusion in the plan:

- Cost and cost-related data to be collected and analyzed.
- Frequency of data collection and analysis.
- Sources of cost-related data.
- Methods of analysis.
- Individuals and organizations involved in the process, along with their responsibilities and duties.
- Limits of acceptable variance between actual costs and the baseline.
- The authority and interaction of the cost control process with the change control process.
- Procedures and responsibilities for dealing with unacceptable cost variances.

6.2.3 Cost Budgeting

Once the costs have been estimated for each WBS task, and all these put together for an overall project cost, a project budget or cost baseline must be constructed. The budget is a spending plan, detailing how and at what rate the project funding will be spent. The budgeting process elements are shown in Figure 6-4. All project activities are not performed at once, resources are finite, and funding will probably be spread out over time. Cost estimates, WBS tasks, resource availability, and expected funding must all be integrated with the project schedule in a plan to apply funds to resources and tasks. Budgeting is a balancing act to ensure the rate of spending closely parallels the resource availability and funding, while not exceeding either. At the same time, task performance schedules must be followed so that all tasks are funded and completed before or by the end of the project schedule.

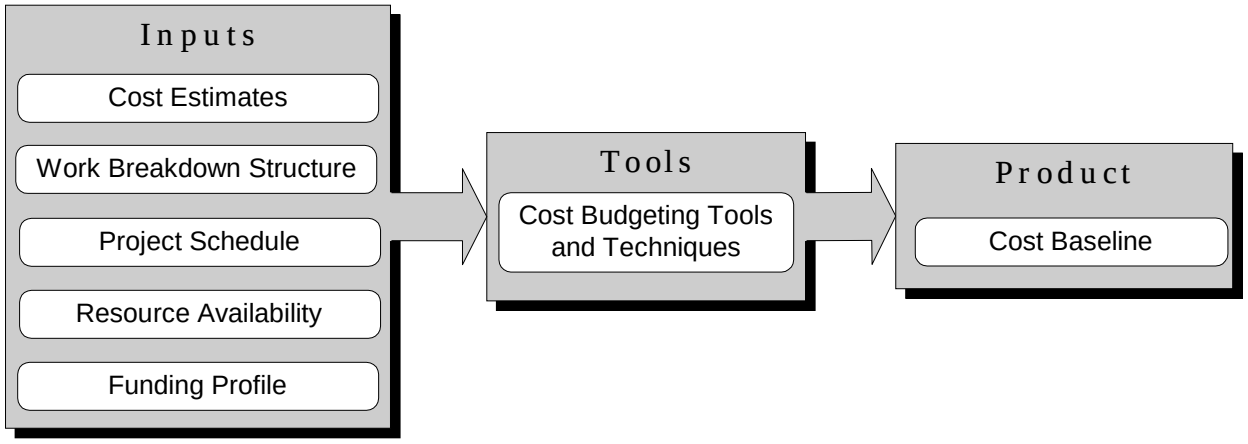

Figure 6-4 Cost Budgeting Elements [1]

The spending plan forms the cost baseline, which will be one of the primary measures of project health and performance. Deviations from this cost baseline are major warning signs requiring management intervention to bring the project back on track.

Various tools and techniques are available to assist in the budgeting process. Most of these are implemented in some form of computer software. Budgeting is usually a major part of project management software.

6.2.4 Cost Control

Cost control is the final step of the cost management process but it continues through the end of the project. It is a major element of project success and consists of efforts to track spending and ensure it stays within the limits of the cost baseline. The following activities make up the cost control process: [1]

- Monitor project spending to ensure it stays within the baseline plan for spending rates and totals.
- When spending varies from the plan determine the cause of variance, remembering that the variance may be a result of incorrect assumptions made when the original cost estimate was developed.
- Change the execution of the project to bring the spending back in line within acceptable limits, or recognize that the original estimate was incorrect, and either obtain additional funding or reduce the scope of the project.
- Prevent unapproved changes to the project and cost baseline.

When it is not possible to maintain the current cost baseline, the cost control process expands to include these activities: [2]

- Manage the process to change the baseline to allow for the new realities of the project (or incorrectly estimated original realities.)
- Accurately record authorized changes in the cost baseline.
- Inform stakeholders of changes.

The input, tool, and product elements of the cost control process are shown in Figure 6-5.

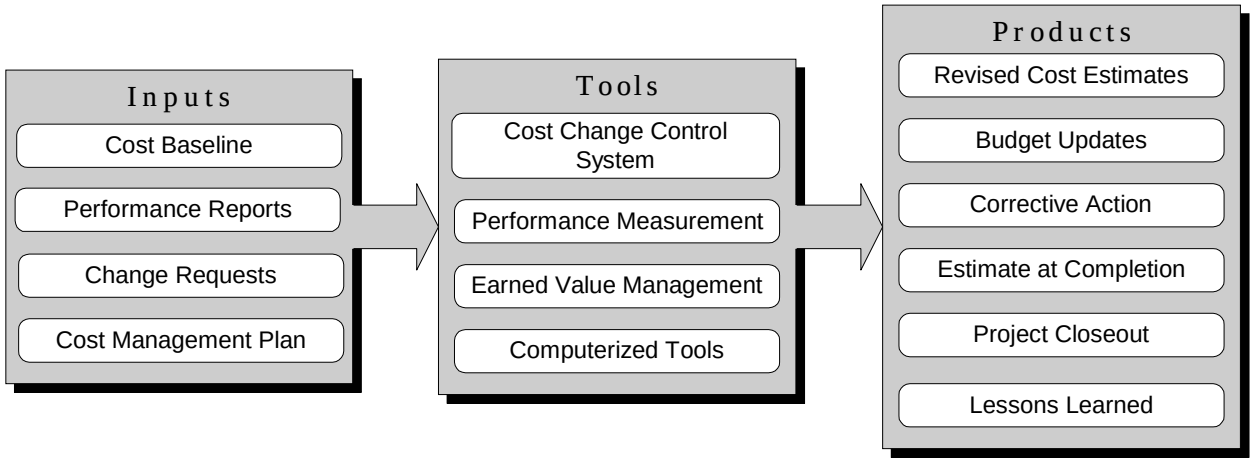

Figure 6-5 Cost Control Elements [1]

The cost control process compares cost performance reports with the cost baseline to detect variances. Guidance on what constitutes unacceptable variance and how to deal with variance can be found in the cost management plan, developed during the estimation activities. Few projects are completed without changes being suggested or requested. All change requests should run the gauntlet of cost control to weigh their advantages against their impact to project costs.

Cost control tools include performance measurement techniques, a working cost change control system, and computer based tools. A powerful technique used with considerable success in projects is **Earned Value Management**, if used appropriately. It requires a fully defined project up front and bottom-up cost estimates, but it can provide accurate and reliable indication of cost performance as early as 15% into the project. [4]

The outputs of cost control includes products which are ongoing throughout the life of the project: revised cost estimates, budget updates, corrective actions, and estimates of what the total project cost will be at completion. Corrective actions can involve anything that incurs cost, or even updating the cost baseline to realign with project realities or changes in scope. Cost data necessary to project closeout are also collected throughout the life of he project and summarized at the end. A final product, extremely important to future efforts, is a compilation of lessons learned during the execution of the project. [1]

6.3 Cost Management Checklist

This checklist is provided as to assist you in cost management. Consider your answers carefully to determine whether you need to examine the situation and take action.

- ❑ 1. Is cost management planning part of your project planning process?
- ❑ 2. Have you established a formal, documented cost management process?
- ❑ 3. Do you have a complete and detailed WBS, including management areas (See Mil-HDBK-881, Appendix H)?
- ❑ 4. Do you have historical information, including costs, from previous similar projects?
- ❑ 5. Have you identified all sources of costs to your project (i.e. different types of labor, materials, supplies, equipment, etc.)?
- ❑ 6. Have you identified proven and applicable estimating methods, models, and/or guides?
- ❑ 7. Have you selected computer software to assist you in estimating, budgeting, tracking, and controlling costs?
- ❑ 8. Do you have justifiable reasons for selecting your methods, models, guides, and software?
- ❑ 9. Are cost issues adequately addressed in your risk management plan?
- ❑ 10. Do you have a working change control process in place?

- ❑ 11. Does the change control process adequately address cost impact?
- ❑ 12. Do your estimates cover all tasks in the WBS?
- ❑ 13. Do you understand your project's funding profile, i.e. how much funding will be provided? At what intervals? How sure is the funding?
- ❑ 14. Have you developed a viable cost baseline that is synchronized with the project schedule and funding profile?
- ❑ 15. Do you have adequate flexibility in the cost baseline?
- ❑ 16. Do you have a plan/process for dealing with variances between cost performance and the baseline?
- ❑ 17. Have you considered incorporating earned value management into your cost management efforts?
- ❑ 18. Are you keeping records of your cost management activity for future efforts?

6.4 References

[1] *Guide to the Project Management Body of Knowledge, A*, Chapter 7, Project Management Institute, 2000.

[2] Baker, Sunny and Kim, *Complete Idiot's Guide to Project Management, 2ed.*, Chapter 15, Alpha Books, 2000.

[3] Chapman, James R., Cost Estimating, 1997, Principle Based Project Management website: www.hyperthot.com/project.htm

[4] Flemming and Koppelman, "Earned Value Project Management, A Powerful Tool For Software Projects," Crosstalk, July 1998: www.stsc.hill.af.mil/crosstalk/1998/jul/value.asp

6.5 Resources

Air Force Cost Analysis Agency (AFCAA): www.saffm.hq.af.mil/afcaa/

Air Force Cost Reference Documents: www.saffm.hq.af.mil/afcaa/reference.html

Cost Tools: www.saffm.hq.af.mil/afcaa/models/models.html

Chapman, James R., Principle Based Project Management website: www.hyperthot.com/project.htm

Constructive Cost Model (COCOMO), information and software, University of Southern California, Center for Software Engineering: http://sunset.usc.edu/research/COCOMOII/index.html

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "Metrics Tools: Software Cost Estimation": www.stsc.hill.af.mil/crosstalk/1995/jun/metrics.asp
- "Cost Realism Methodology for Software-Intensive Source Selection Activities": www.stsc.hill.af.mil/crosstalk/1995/jun/cost.asp
- "Earned Value Project Management": www.stsc.hill.af.mil/crosstalk/1998/jul/value.asp
- "Pattern-Based Architecture: Bridging Software Reuse and Cost Management": www.stsc.hill.af.mil/crosstalk/1995/mar/pattern.asp
- "Does Calibration Improve Predictive Accuracy": www.stsc.hill.af.mil/crosstalk/2000/apr/ferens.asp
- "Project Recovery… It Can be Done": www.stsc.hill.af.mil/crosstalk/2002/jan/lipke.asp
- "Driving Quality Through Parametrics": www.stsc.hill.af.mil/crosstalk/1998/nov/galorath.asp
- "Future Trends, Implications in Cost Estimation Models": www.stsc.hill.af.mil/crosstalk/2000/apr/boehm.asp
- Practical Software Measurement, Performance-Based Earned Value": www.stsc.hill.af.mil/crosstalk/2001/sep/solomon.asp
- "New Air Force Software Metrics Policy": www.stsc.hill.af.mil/crosstalk/1994/apr/xt94d04a.asp
- "Statistical Process Control Meets Earned Value": www.stsc.hill.af.mil/crosstalk/2000/jun/lipke.asp

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, Chapter 13, OO-ALC/TISE, May 2000. Download at: www.stsc.hill.af.mil/gsam/guid.asp

MIL-HDBK-881, Work Breakdown Structure:
www.acq.osd.mil/pm/newpolicy/wbs/mil_hdbk_881/mil_hdbk_881.htm
http://web2.deskbook.osd.mil/reflib/DDOD/003EH/001/003EH001DOC.HTM

NASA, *Parametric Cost Estimating Handbook, 2 ed*. Online version: www.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm
Download and print version: www.jsc.nasa.gov/bu2/NCEH/index.htm

Parametric Estimating Handbook:
http://web2.deskbook.osd.mil/reflib/DDOD/005EV/001/005EV001DOC.HTM#T2

Practical Software and Systems Measurement Support Center: www.psmsc.com

Software Cost Estimation Website: www.ecfc.u-net.com/cost/index.htm

Software Technology Support Center Course: *Life Cycle Software Project Management*, Estimation, earned value, etc., 9 October 2001.

Xanadu, Slides on software project estimation:
http://xanadu.bmth.ac.uk/staff/kphalp/students/bsi/predict/tsld002.htm

Chapter 7

Time and Schedule Management

CONTENTS

This page intentionally left blank.

Chapter 7

Time and Schedule Management

7.1 Introduction

Time is a resource that is constantly in use and, once spent, cannot be recovered. While there may be an endless supply of days, they are delivered at a constant rate and cannot be compressed or expanded. Additional time, if any is available, can only be bought by giving up something else. Time is also money. Resources, especially people, cannot be used over time without paying for them. Because we all use the same time line, projects must conform to the schedule needs of the bigger picture if they are to be of any value. New fighter aircraft are only valuable if they are delivered early enough to provide air superiority, before the enemy can field a better aircraft. Time and schedule management is absolutely essential for project success. Running out of time, like running out of money, will bring your project to a halt.

Cost, schedule, scope, and quality are four attributes of all development projects. The project manager can at best control any three of these four attributes. If the capabilities and performance (scope), development cost, and quality are defined, schedule is pre-determined. When schedule is fixed, one or more of the other three attributes must become variables that change to meet schedule requirements. Each of the attributes is bounded and none can have a zero value. Software cannot be produced in less than a minimum time depending on size, complexity, and environment. Project scope must include a minimum set of capabilities. Projects that stay within predicted schedule are the exception, not the rule. A project manager who can control schedule while achieving performance, quality, and cost goals has learned the secrets of proper planning and execution.

The first step in controlling a schedule is "knowing" what the schedule is. Three other primary factors are also essential to schedule control: managing project costs, system scope, and quality. In general, if system capabilities, project costs, and product quality are well defined and controlled, the schedule can be realistically predicted using appropriate estimating methods, and then followed. While schedules slip out of control for a number of reasons, foremost among those reasons are: (1) the schedule was significantly under-estimated, and (2) the scope (size) increased during development. Success is easier to achieve when scope, costs, and quality are known up front and controlled. Schedule is used to monitor project performance, but it must be remembered that management of the project determines schedule and not vice versa.

Managing time and schedule involves determining what needs to be done, in what order, estimating how long it will take, scheduling tasks to coincide with resource availability, tracking project progress with respect to the schedule, and taking preventive or corrective action when the project begins to deviate from the schedule. It is a logical step in planning a project, and runs throughout the project. It can be viewed as a separate discipline in its own right, apart from project management. Some project managers work with a scheduler assistant whose sole purpose is to coordinate, document, and monitor the project schedule.

This chapter summarizes the process of schedule development and control, as well as providing an overview of methods and tools used in the process.

7.2 Process Description

The time management process consists of the five activities shown in Figure 7-1. The first four are project schedule preparation steps and are usually performed during the planning phase. The last activity involves monitoring and tracking the project performance with respect to the schedule, and implementing corrective actions when schedule performance varies significantly from the planned schedule. The schedule control activity is performed throughout the remainder of the project.

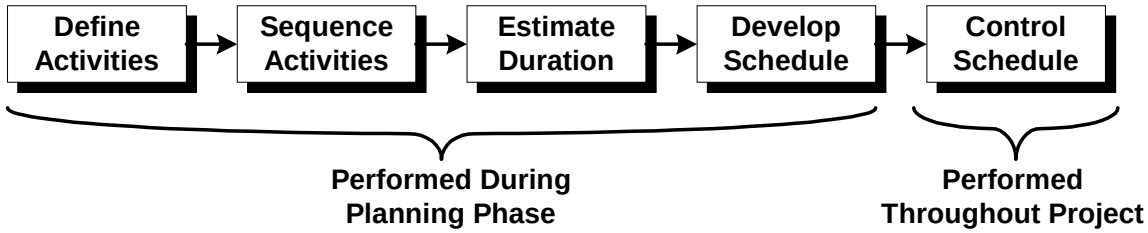

Figure 7-1 Time Management Process [1]

Schedule development is illustrated in greater detail in the example in Figure 7-2. In the first step, all those activities needed to accomplish the project goals are identified and defined. In this simplified example there are only eight activities. The second step involves determining which activities must be performed before others and sequencing them to match this dependence. In the next step the duration of each activity is estimated. Finally, the activities are put into a time frame, properly sequenced, with parallel activities matching the availability of resources and manpower. When this is done there will be a single string of dependent activities which will collectively have the longest duration. This string is called the *critical path*. All other activities depend on this sequence of activities. If an activity on the critical path is delayed, the project is delayed and the schedule lengthened. Identifying the critical path and monitoring its activities are essential to successful schedule control. Each of these process steps is explained in greater detail in the following paragraphs, along with schedule control.

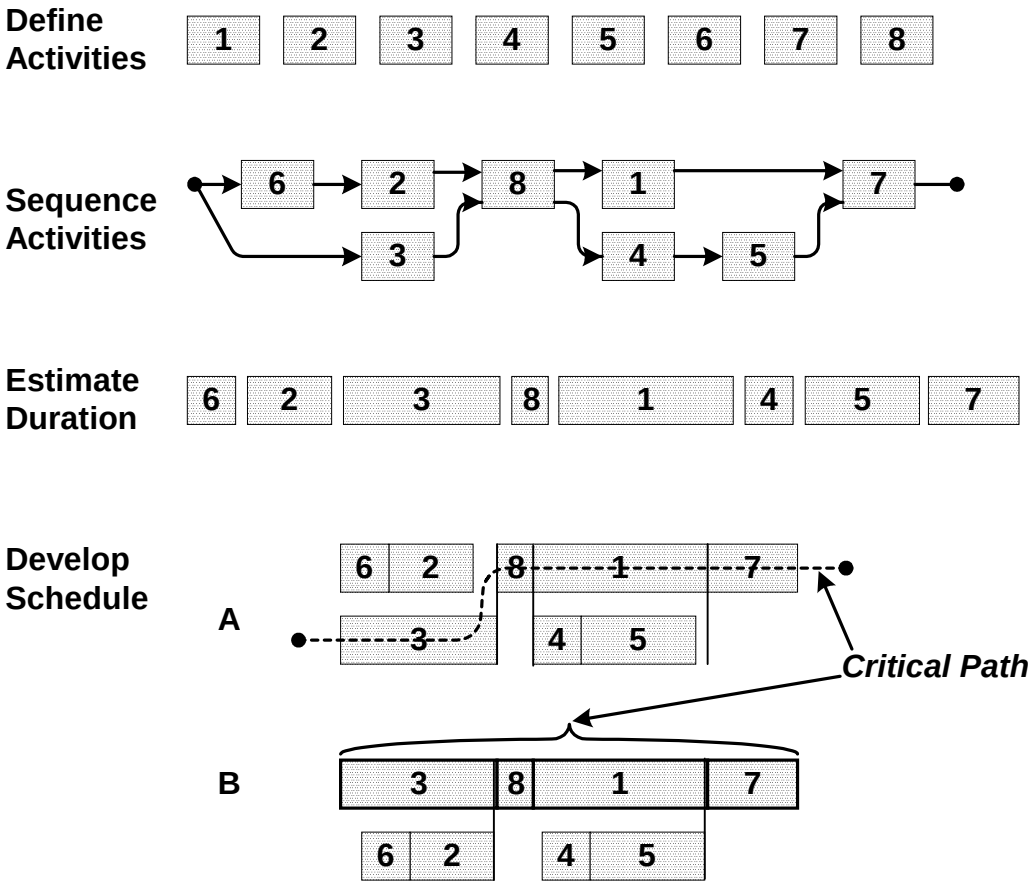

Figure 7-2 Schedule Development Overview

7.2.1 Activity Definition

Activity definition is the process of identifying all the activities needed to produce the project deliverables defined in the Work Breakdown Structure (WBS). (The WBS is discussed in Chapter 3, Section 3.2.1.3.) Deliverables should have appropriate and sufficient activities associated with them to accomplish all project objectives. [1]

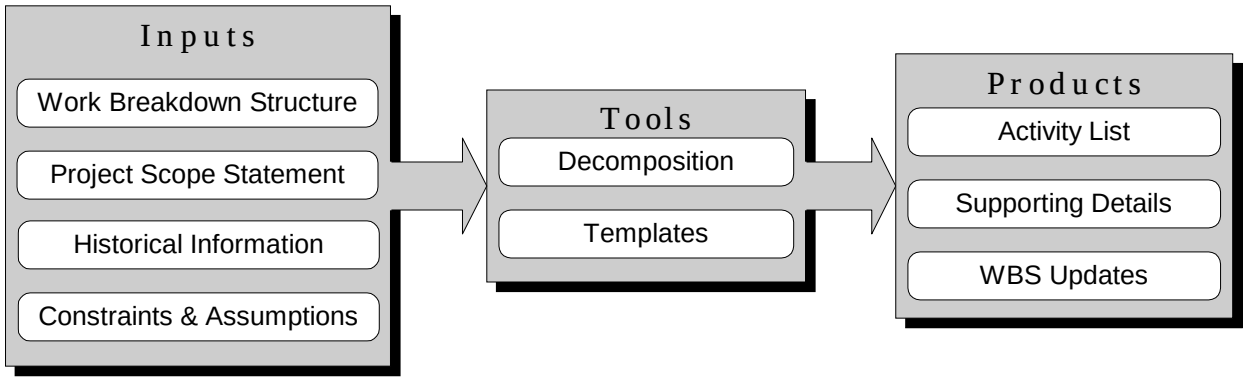

Figure 7-3 Activity Definition Elements [1]

Figure 7-3 lists the various elements associated with activity definition. The primary input is the WBS. Other inputs include the project scope statement to ensure all activities fall within project scope, historical information from previous projects to see how it was done in the past, and any constraints or assumptions affecting project activities.

Activity definition tools include decomposition and templates. Decomposition is the process of dividing and subdividing tasks into smaller, more manageable components. Remember, activity definition decomposition is dealing with activities and not with deliverables. Templates are prepared, often from previous project activity lists, to ensure all necessary information is documented. They should include blanks for such things as needed skills, hours of effort, materials, risks, deliverables, etc. [1]

The primary product of activity definition is the activity list, describing each activity to be performed. The activity list is accompanied by detailed information on how the activities were defined, e.g. the decomposition method used, etc. In addition to creating the activity list, the intense scrutiny of the WBS by the project team usually illuminates inconsistencies or omissions in the WBS, leading to updates to the WBS.

7.2.2 Activity Sequencing

Activity sequencing is the process of determining dependencies between activities. Each activity is analyzed to determine what other activities must be performed before it can begin. The result is a network of activity chains like that shown in Figure 7-4. There various forms of these networks and collectively they are called *Project Network Diagrams*.

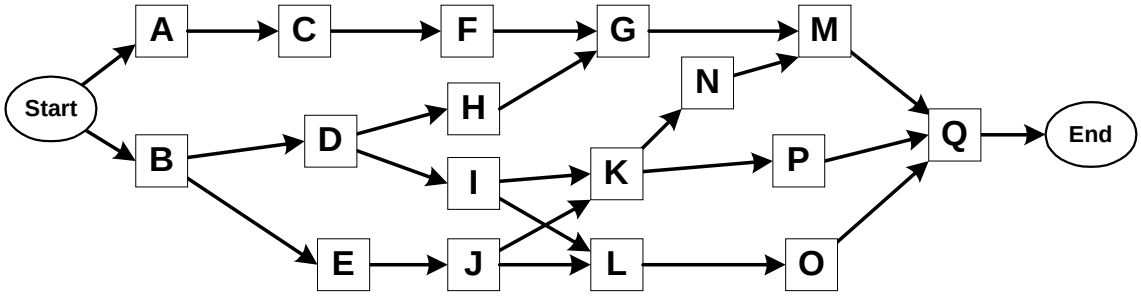

Figure 7-4 Activity Network Diagram (Precedence Diagramming Method)

The elements of the sequencing process are shown in Figure 7-5. The primary input is the activity list developed during activity definition. These are the activities that must be sequenced. The project product definition, lists of dependencies, and expected projected milestones are used to ensure the sequencing process takes all issues and requirements into consideration. Dependencies include the following types: [1]

- Mandatory dependencies – physical limitations, flow of work, and obvious sequences of events. Often called hard logic.
- Discretionary dependencies – defined and documented by the project team, best practices, and preferred logic.
- External dependencies – relationships of the project with external (to the project) activities. An example would be flight-testing an avionics upgrade. It is dependent upon the availability of aircraft, pilots, weather, etc.

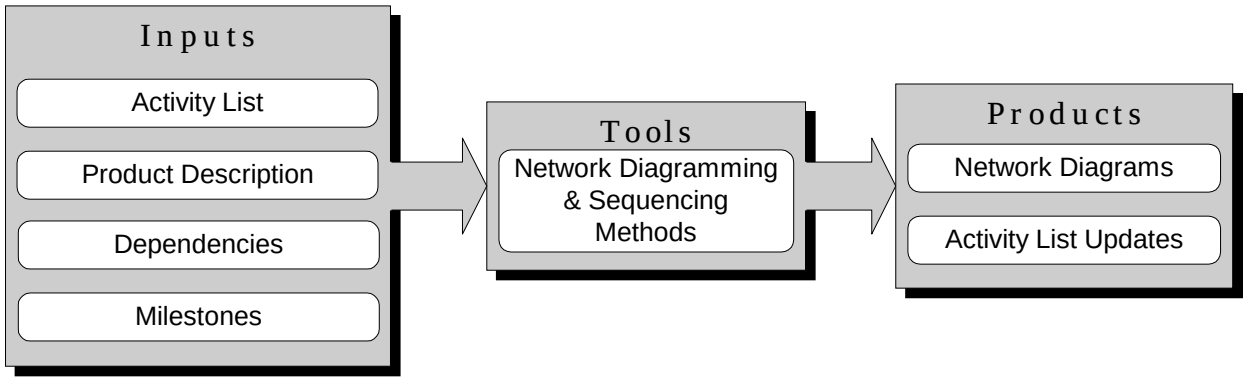

Figure 7-5 Activity Definition Elements [1]

Various methods have been developed for activity sequencing. Two of the most common are the Precedence Diagramming Method (PDM), also known as Activity-on-Node (AON), and the Arrow Diagramming Method (ADM). AON is implemented in most project management software but can be also be performed manually. The diagram uses boxes to represent activities and arrows to represent dependencies, similar to Figure 7-4. It is based on four types of precedence relationships: [1]

- Finish-to-start – the successor activity cannot start until the predecessor activity has finished. This is the most common relationship.
- Finish-to-finish – the successor cannot finish work until the predecessor has finished.
- Start-to-start – the successor cannot start work until the predecessor has started.
- Start-to-finish - the successor cannot finish work until the predecessor has started.

ADM, also known as Activity-on-Arrow (AOA), uses arrows to represent activities, and uses nodes to show dependencies between the activities, as shown in Figure 7-6. Only finish-to-start dependencies are used for ADM.

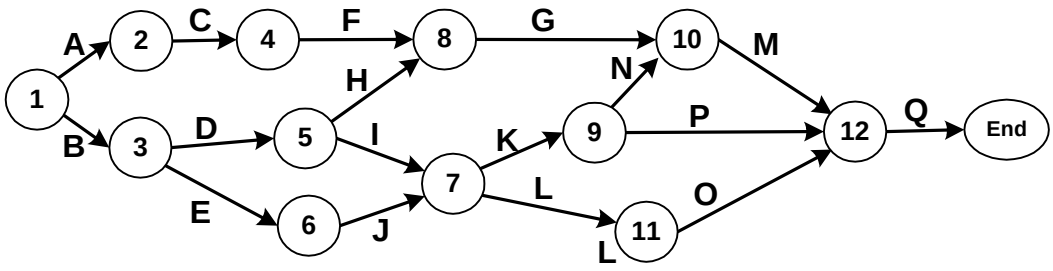

Figure 7-6 Activity Network Diagram (Arrow Diagramming Method)

Neither PDM nor ADM allow loops or conditional branches (if-then-else). If that type of diagramming is needed, *conditional diagramming* methods can be used. Two examples are Graphical Evaluation and Review Technique (GERT) and System Dynamics models. [1] More on diagramming methods can be found in 7.5, Resources.

7.2.3 Estimating Activity Duration

With activities defined and sequenced, the next step in developing a schedule is estimating the duration of each activity. The elements of this activity are shown in Figure 7-7 and include various inputs, tools and outputs.

When estimating, the activity list is used to provide details on each activity and to ensure all activities are estimated. A knowledge of how an activity can and should be performed, and what is required to do it is essential. Estimators must know about resource capabilities and availability, along with project constraints (funding, people, etc.), and assumptions. Some activities' durations can be shortened by putting more people to work on them, where others cannot. Nine women can't produce a baby in one month. Even if nine people can accomplish a task in less time, there may not be nine people available to do it, or it may be cost prohibitive. One of the greatest helps to estimating duration is historical information from previous projects. Seeing what actually happened in the past can be a good reality check for current estimates. Risks can have a significant impact on activity duration and may be looked at as either threats or opportunities. [1]

7.2.3.1 Expert Judgment and Analogous Estimating

The tools used in estimating are varied and are all likely to be used on the same project. Some activities can best be estimated using one method, some another. *Expert judgment* involves the use of people who have an expert understanding of the activity and are guided by historical information. In *analogous estimating*, also known as top-down estimating, a previous similar activity is used as a model for estimating future duration. It is a form of expert judgment and requires that the previous activity be truly similar to be accurate.

7.2.3.2 Quantitative Estimating

Quantitative estimating involves determining the quantity of work to be accomplished for each activity. This could include writing a certain number of lines of code, designing a certain numbers of circuit boards with specific levels of complexity, processing a specific number of pages, producing a specific quantity of engineering drawings, etc. A unit-rate, the rate of time to accomplish a unit of work, is divided into the quantity to produce a duration for the specific activity. Unit rates are derived from historical data or from industry standards or models. In addition to the specific duration estimates, reserve time should be added to provide a buffer against contingencies which may arise from realized risks.

Using this method for software estimating requires the development environment (people, development system, process), as well as product scope and application, to be essentially identical to the projects used to produce the unit rate. Humans tend to develop software estimates that are optimistic: that is, success-oriented. Care must be taken to assure that the estimates are realistic and within organization historical data bounds.

7.2.3.3 Parametric Estimating

Parametric estimating uses mathematical models, cost estimating relationships (CERs) and rules of thumb to realistically estimate software activity schedules. CERs are relationships between schedule, cost and product size (scope) to arrive at realistic schedules. Parametric models are derived from historic data and allow the estimator to incorporate environment and product information, as well as project constraints, into the activity characteristics. Parametric estimating is usually easier and faster than quantitative methods, but the method is only accurate if the correct model or CER is used in the appropriate manner.

7.2.3.4 Computer Tools

Computer tools are used extensively, especially with parametric estimating, to assist in schedule or activity duration estimating. The tools range from simple spreadsheets to project management software for specialized system, hardware, and software schedule estimates. Computer tools speed the estimation process, reduce the incidence of errors caused by optimism and calculation, and allow for consideration of process alternatives. Widely used hardware estimating tools include Price-H and SEER-H. Widely used software estimating tools include COCOMO II, Price-S, Sage, SEER-SEM, and SLIM. More information about these tools can be found in section 7.5, Resources.

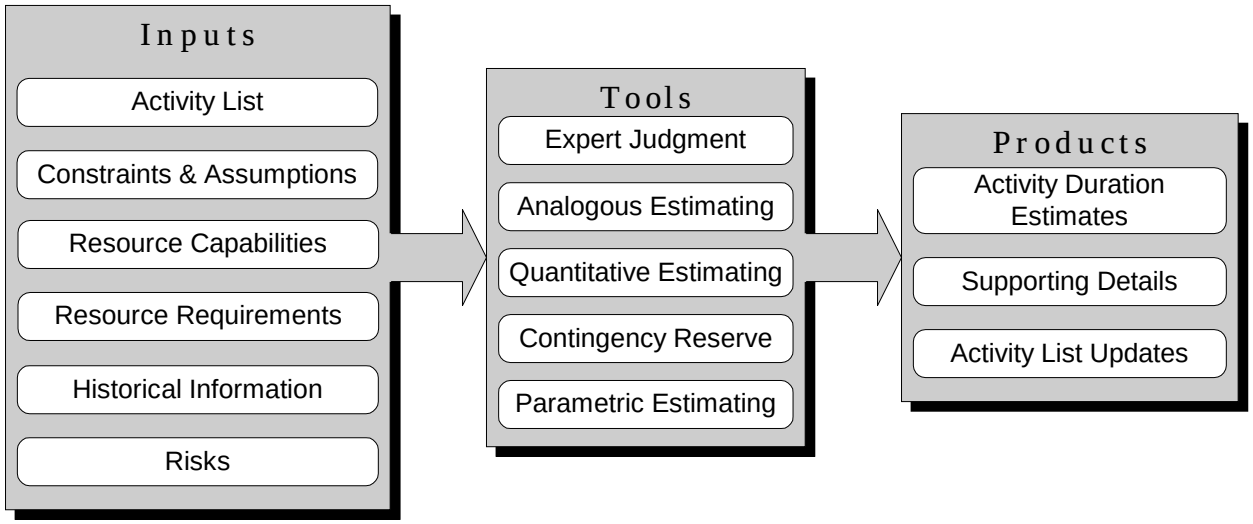

Figure 7-7 Activity Duration Estimation Elements [1]

The output from activity duration estimating includes the estimates themselves, and details explaining the estimation methods used and reasons for choosing them. After working with each activity in greater detail, there will probably be changes to the activity list.

7.2.4 Schedule Development

7.2.4.1 Inputs

The schedule development process combines the activity sequence from the project network diagrams with the duration estimates to build chains of activities, the basis for the project schedule. This is usually done with the help of project management software to ensure calculations are correct and keep track of all activities. Most project management software will also produce charts to help visualize, track, and control the project schedule. Another thing they do is force the user to provide essential project information and follow good project planning processes.

If activity sequence and duration were all that was necessary, the job would fairly simple. Far more is needed to develop a schedule. The resources needed to perform the work, and the capabilities and availability of resources, including skilled people, materials, equipment, tools, facilities, etc. have a major impact on when activities can start and which can run simultaneously. Other significant inputs to schedule development are the project constraints, assumptions, and the calendar. Holidays, weekends, and vacations must be considered. Lead and lag times of materials and equipment must be built into the schedule. For example, some electronic parts are long lead items and must be ordered weeks or even months before they are delivered. New equipment will probably need to be inspected, installed and tested, necessitating a lag time between delivery and actual availability for use. Necessary training time for personnel must be considered. Activity attributes such as when, where, or how the activity must be performed will affect the schedule. If something cannot be performed in cold weather or if it must be performed at a specific location, the scheduler will need to consider the seasons or include transportation time. As always, project risks must be figured into the schedule. These inputs to schedule development are shown in Figure 7-8.

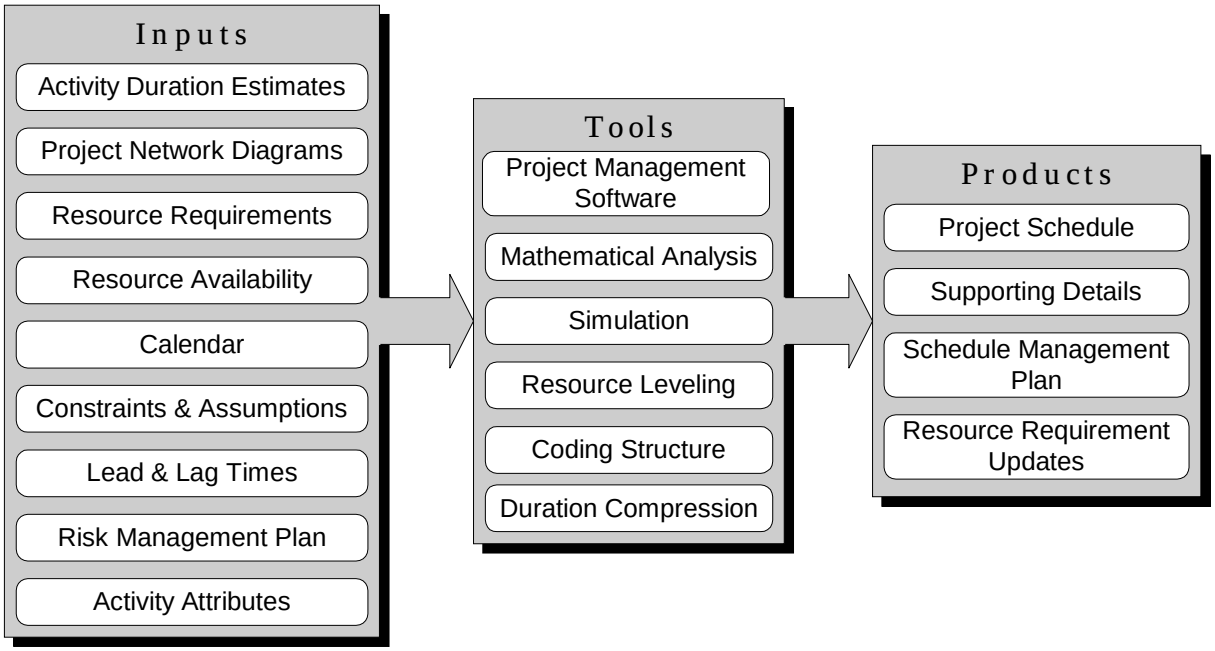

Figure 7-8 Schedule Development Elements [1]

7.2.4.2 Tools

The most important tool for schedule development is common sense. *Project management software* can take the drudgery out of schedule development. The better packages can guide users in gathering and properly using pertinent information. However, to fully utilize the capabilities of the software, the user should be properly trained, have an understanding of project management processes and techniques, and exercise his or her common sense.

Beyond good software, there exists a host of techniques and methods to help build an efficient, optimized schedule. One of the more widely used scheduling tools is *mathematical analysis*, which consists of calculating the range of possible start and stop dates for each activity. This shows when activities could theoretically start and how long they could take to accomplish. To this indefinite schedule must be added resource capabilities and availability, as well as the other considerations and constraints listed as schedule inputs in Figure 7-8. This additional information allows the schedulers to finalize activity start and stop dates. Once this is complete, the chain of dependent activities which has the longest total duration is identified and designated as the *critical path*. Any activity along this path that starts late or takes longer than planned lengthens the whole project schedule. The most commonly used implementations of mathematical analysis are *Critical Path Method (CPM), Program Evaluation and Review Technique (PERT),* and *Graphical Evaluation and Review Technique (GERT).*

Other tools may be used instead of, or in concert with, the above-mentioned mathematical analyses. The following categories are worth mentioning: [1]

- *Simulation* – Performing computer simulations of the schedule, and what-if analyses to determine the best schedule.
- *Resource Leveling* – Basing or adjusting the schedule to desired or expected levels of manpower and/or other resources.
- *Coding Structure* – Method of coding or labeling activities to indicate attributes which may be used in grouping or sorting the activities into more logical or useful sequences.
- *Duration Compression* – Methods of determining ways to shorten the schedule without reducing the project scope.

7.2.4.3 Products

The primary output of the scheduling effort is the project schedule. This includes planned start and finish dates for each activity and usually consists of a top-level summary, or *master schedule*, and a more detailed version. Schedules may be presented in a tabular format but most people prefer some type of graphical format because it makes it easier to see and understand the overall picture. Project network diagrams with the dates added may be used to show program logic and the critical path. Another type of schedule chart is the bar chart also known as the Gantt chart. Activity start, finish, and duration are represented as bars on time graph background, shown in Figure 7-9.

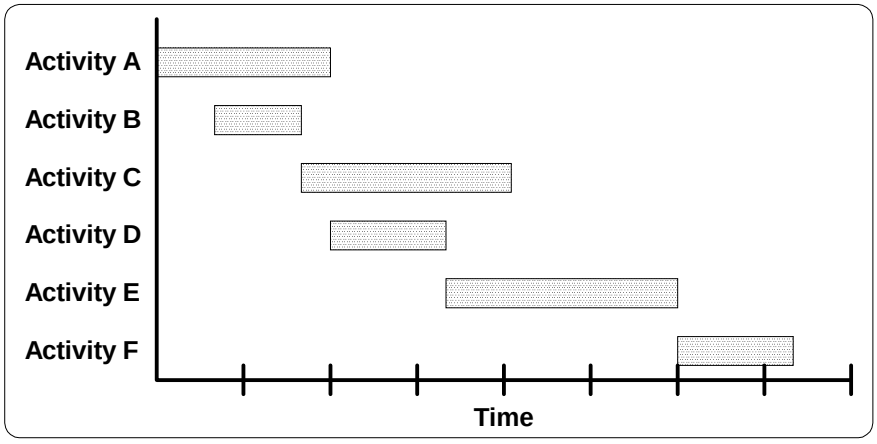

Figure 7-9 Example Gantt Chart

A third type of schedule chart is the milestone chart. This chart lists project milestones on the left and their planned and actual completion dates on a time graph background, shown in Figure 7-10. The various types of charts have different levels of details and are meant for different audiences.

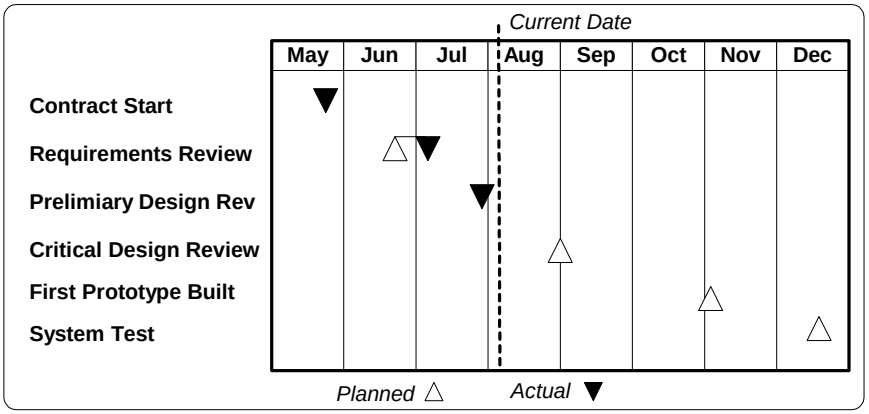

Figure 7-10 Example Milestone Chart

The other essential product of schedule development is the *Schedule Management Plan*. This plan describes the process to be used to track the schedule and identifies what will be measured and tracked to determine whether the actual performance is following the planned schedule. It also defines how the schedule will be updated and changes may be made. This plan may be formal or informal, detailed or top-level, according to the needs of the project.

The two other outputs of the schedule development process are the perennial supporting details, how the schedule was developed and why those methods were used, and updates to the resource requirements. Now that the schedule is fixed on the calendar, there will need to be a final coordination between the schedule and available resources. Until this is finalized, the schedule is tentative.

7.2.5 Schedule Control

The primary method of schedule control is to compare actual schedule performance reports with the planned schedule and take corrective action when there is a danger of falling behind schedule along the critical path. The inputs to the schedule control process are shown in Figure 7-11. In addition to the project schedule, the schedule management plan, and regular performance reports, there will probably be change requests. For example, someone may request an extension on the duration of a particular activity because ordered parts will not be delivered in time to complete the activity on schedule. If the extension does not impact the critical path, it may be sufficient to just allow the extension and watch the situation closely. If it does impact the critical path, corrective action must be taken to get the parts earlier, make up for the lost time by compressing other activity on the critical path, or lengthening the project schedule.

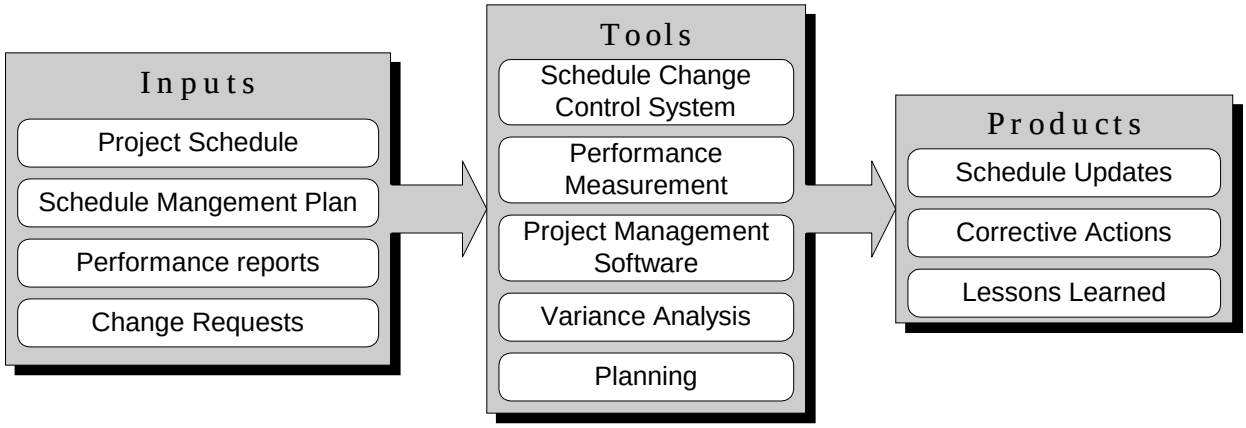

Figure 7-11 Schedule Control Elements [1]

Controlling the schedule requires careful, constant monitoring of the project status. Anything that can impact the schedule, slow down, or postpone an activity must be watched. This includes changes in requirements. A change in scope will usually require changes in budget and in schedule. Specific areas to monitor for each activity are:

Before the Activity

- Preparatory actions: permissions, paperwork, clearances, etc.
- Equipment deliveries and setup
- Long-lead material acquisitions
- Availability of manpower and other resources

During the Activity

- Actual progress of the activity – Talk doesn't count
- How much of the activity is left to be accomplished
- Activity milestones, if applicable
- Any problems that may slow or postpone work, whether technical, budgetary, resources, etc.
- Other projects competing for your resources

Regular performance reports of progress, problems, risks, and plans for upcoming and ongoing activities will provide the oversight necessary to track schedule performance. Informal monitoring, outside the briefing room, among the team leaders and workers will provide insight to events and issues that may become problems down the road. The sooner a potential problem can be detected the easier it is to avoid it or reduce its effects. Bringing project schedule back on track usually involves sacrifices in the other project attributes, quality and scope. Increasing project resources invariably decreases the possibility of schedule improvement, as stated in Brooks' Law: *"Adding manpower to a late software project makes it later."* [3]

When actual progress deviates from the planned schedule *variance analysis* is used to determine how much difference there is and what impact the variance will have. If tasks are not on the critical path, there should be some flexibility in when they are performed. This range of time for their performance is called *float* time. The amount of float a particular activity has will, to a large extent, determine your response to schedule variance for that activity. Critical path activities have no float. The schedule change control system, defined in the schedule management plan, should

provide a standard process and guidelines for dealing with schedule impacts and changes. When threatened with schedule impacts, i.e. if float time is running out or if a critical path activity is delayed, you will have to carefully evaluate your resources and use creativity to solve or get around the problem. Remember, pilots aren't in trouble unless they run out of airspeed, altitude, and ideas at the same time. The following methods are often used to deal with schedule problems.

Reserve Hopefully, you've reserved some time for just such occasions and can use some now. *Star Trek's* Scotty asked how he could keep his reputation as a miracle worker if he didn't tell people things would take longer than he thought.

Substitution While waiting for unforeseen delays, do other tasks or parts of tasks that can be done now.

Preparation Prepare everything for this and follow on activities so that startup and transition times can be reduced.

Miracles Consult with your experts to see if there is a way to catch up by working faster and smarter, or improving the process.

Manpower Sometimes putting more people on a task is a viable way to speed it up. If additional manpower can be found they may be able to overcome a delay. However, adding manpower is usually not a good solution (See Brooks' Law). Don't forget the additional budget costs you may incur, or the time needed to come up to speed on the project, or the initial additional delay caused by diverting project personnel to bring added personnel up to speed.

Overtime Nights, weekends, and holidays can sometimes be used to overcome a delay. However, this method does not come without high costs in morale, quality, etc. Use this method sparingly.

Diligent monitoring, creative problem solving, being proactive instead of reactive, and carefully preparing the schedule in the first place go a long way toward keeping the project on schedule. There will always be unforeseen events and problems because of chance, inexperience, or incomplete information. For those problems that cannot be solved within the bounds of the current schedule, extension of the schedule will be necessary. "Slipping" the schedule is also a method of dealing with project time problems and is not necessarily catastrophic event. However, continually asking for more time, like continually asking for more money, will quickly diminish support for the project and confidence in the project manager.

The outputs of the project control process will be schedule updates and, corrective actions, and lessons which can be applied later in the current project and on future projects.

7.3 Time and Schedule Checklist

This checklist is provided as to assist you in Time and Schedule Management. Consider your answers carefully to determine whether you need to carefully examine the situation and take action.

7.3.1 Preparing for Schedule Development

- ❑ 1. Have you identified an experienced, knowledgeable team to develop the schedule?
- ❑ 2. Has a process to develop the project schedule been defined and documented?
- ❑ 3. Are all time and date requirements for the project known and documented? (What is needed when?)
- ❑ 4. Are there unreasonable time constraints for the project?
- ❑ 5. Are resource capabilities and availability known?
- ❑ 6. Do you have the project constraints, assumptions, and risk plan documented?
- ❑ 7. Do all deliverables listed in the WBS have adequate and appropriate activities identified to produce them?
- ❑ 8. Have you chosen an appropriate project management software package, and are you experienced or have you been trained in using it?

- ❑ 9. Is historical duration data available for project activities?

7.3.2 Schedule Development

- ❑ 10. Have you identified appropriate methods and models for estimating activity duration?
- ❑ 11. Have all activities been sequenced by putting them into an activity network and indicating the dependencies between them?
- ❑ 12. Have durations been estimated for all activities?
- ❑ 13. Have the activity durations been reviewed by people experienced in those activities?
- ❑ 14. Has the critical path been identified?
- ❑ 15. Has float time been documented for all activities not on the critical path?
- ❑ 16. When developing the schedule, are you using resource leveling and remembering holidays, vacations, and sick time?
- ❑ 17. Have you developed and documented a reality-based schedule?
- ❑ 18. Have you built a time reserve into your schedule for contingencies and unforeseen events?
- ❑ 19. Has your schedule been entered into a program management software package?

7.3.3 Schedule Control

- ❑ 20. Have you developed and documented a schedule management plan?
- ❑ 21. Do you know how, what, when, why, and how much to monitor for schedule control?
- ❑ 22. Are you being proactive vs. reactive in your approach to schedule control by looking ahead and asking what could go wrong?
- ❑ 23. Do you have a schedule change process documented and implemented?
- ❑ 24. Are you monitoring all preparatory actions, acquisitions, deliveries, and resources for each activity to make sure they are all complete and ready when it is time to begin the activity?
- ❑ 25. Are you using experienced people to make and review schedule progress reports?
- ❑ 26. Have the various groups and individuals been given sufficient levels of responsibility, accountability, and authority to perform their tasks? Have they agreed to their assigned roles?
- ❑ 27. Are you employing regular formal and informal schedule monitoring and progress reports
- ❑ 28. Are you constantly aware of project milestones and your schedule progress?
- ❑ 29. Are you solving schedule problems by being creative and using common sense vs. extending the schedule?
- ❑ 30. Are you documenting your progress, problems, issues, solutions, and lessons learned?

7.4 References

[1] *Guide to the Project Management Body of Knowledge, A*, Chapter 6, Project Management Institute, 2000. Download complete PMBOK 1996 or PMBOK 2000 preview at: www.hunu.edu.cn/pmrc/download.htm

[2] *NASA Systems Engineering Handbook*, Chapter 4, June 1995. Download PDF version at: http://ldcm.gsfc.nasa.gov/library/library.htm

[3] Brooks, F. P., Jr., *Mythical Man-Month: Essays on Software Engineering*, (Reading, MA: Addison Wesley, Inc.), 1975, p 25.

7.5 Resources

Builders Net, Critical Path Method (CPM) tutorial: www.buildersnet.org/cpmtutor/

Critical Path Method Tutor, U.S. Army. Copy files: ftp://www.buildersnet.org/CPMTeach/

Guide to the Project Management Body of Knowledge, A, Chapter 6, Project Management Institute, 2000.

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "Quantitative Software Management - Software Cost and Schedule Estimation": www.stsc.hill.af.mil/crosstalk/1996/oct/xt96d10h.asp
- "Applying Management Reserve to Software Project Management": www.stsc.hill.af.mil/crosstalk/1999/mar/lipke.asp
- "Project Scheduling According to Dr. Goldratt": www.stsc.hill.af.mil/crosstalk/2001/jan/perkins.asp
- "Metrics Tools: Effort and Schedule": www.stsc.hill.af.mil/crosstalk/1995/mar/metrics.asp
- "Project Recovery … It Can Be Done": www.stsc.hill.af.mil/crosstalk/2002/jan/lipke.asp
- "New Air Force Software Metrics Policy": www.stsc.hill.af.mil/crosstalk/1994/apr/xt94d04a.asp

Illinois Institute of Technology, Construction Planning and Scheduling class notes.

Gantt Chart notes: www.iit.edu/~jshi/courses/CAE471_L2.PDF

Critical Path Method Schedules notes: www.iit.edu/~jshi/courses/CAE471_L3.PDF

Arrow Diagramming Method notes: www.iit.edu/~jshi/courses/CAE471_L4.PDF

Precedence Diagramming Method notes: www.iit.edu/~jshi/courses/CAE471_L5.PDF

Resource Allocation notes: www.iit.edu/~jshi/courses/CAE471_L7.PDF

Time – Cost Tradeoff notes: www.iit.edu/~jshi/courses/CAE471_L8.PDF

Learning Factory, MS Project scheduling tutorial: http://lfserver.lf.psu.edu/lf/msproject/over.htm

Manchester Metropolitan University, Scheduling tutorial: www.doc.mmu.ac.uk/online/SAD/T04/projman.htm

Smartdraw scheduling tutorials: www.smartdraw.com/resources/centers/gantt/resources.htm

Software Estimating Tools and Methods:

- Constructive Cost Model (COCOMO II), information and software, University of Southern California, Center for Software Engineering: http://sunset.usc.edu/research/COCOMOII/index.html
- Price-H, information and software: www.pricesystems.com
- Sage, information and software: www.seisage.com
- SEER-H, information and software: www.galorath.com
- SLIM, information: www.jsc.nasa.gov/bu2/PCEHHTML/pceh.htm , www.qsm.com/THRU_PUT.pdf , www.qsm-uk.com/course.pdf

Software Technology Support Center Course: *Life Cycle Software Project Management*, Estimation, earned value, etc., 9 October 2001.

Univ. of Minnesota, Tutorial on scheduling, Gantt charts: www.me.umn.edu/courses/me4054/assignments/gantt.html

Chapter 8

Measurement and Metrics

CONTENTS

This page intentionally left blank.

Chapter 8

Measurement and Metrics

"You cannot control what you cannot measure." - Tom DeMarco, 1982.

Imagine going on a road trip of over a thousand miles. This is easy because most of us have done the real thing several times. Now imagine that your car has no speedometer, no odometer, no fuel gauge, no temperature indicator. Imagine also that someone has removed the mile markers and road signs from all the roads between you and your destination. And just to complete the experiment, remove your watch.

What was once a simple journey becomes an endless series of guesses, fraught with risks. How do you know where you are, how far you've gone, how far to go, when to gas the car, should you stop here or try to make the next town before nightfall. You could break down, run out of gas, be stranded, take the wrong road, bypass your destination, or waste time trying to find where you are and how to reach your destination. Clearly, some method of measuring certain indicators of progress is essential for achieving a goal.

Imagine again going on a road trip. This time the cockpit of the car is filled with instruments. In addition to what you have been accustomed to in the past, there is now speed in feet and yards per second, and as a percentage of c (light speed), oil pressure in millibars, estimated time to deplete or recharge the battery, fuel burn rate, fuel weight, oil viscosity and transparency indicators, antifreeze temperature and pressure, engine efficiency, air conditioning system parameters (pressures, temperatures, efficiency), elevation, rate of climb, heading, accelerometers for all directions, indicators for distance and time to destination and from origin, and inside air temperatures for eight different locations in the car. There are instruments to count how many cars pass, vibration levels, and sound pressure levels within and outside the car. There are weather indicators for outside temperature, humidity, visibility, cloud ceiling, ambient light level, true and relative wind speeds and directions, warning indicators for approaching storms and seismic activity, etc. Along the roads will be markers for every hundredth mile and signs announcing exits every quarter mile for five miles before an exit is reached. Speed changes will be announced by signs in 5 MPH increments. To some this may seem like a dream come true, at least the cockpit part. But careful consideration will soon reveal that the driver will be inundated and quickly overwhelmed with unnecessary, confusing data. Measuring things, in itself, is no prescription for achieving a goal. It can even make the goal unattainable.

8.1 Introduction

Metrics are measurements of different aspects of an endeavor that help us determine whether or not we are progressing toward the goal of that endeavor. They are used extensively as management tools to provide some calculated, observable basis for making decisions. Some common metrics for projects include schedule deviation, remaining budget and expenditure rate, presence or absence of specific types of problems, and milestones achieved. Without some way to accurately track budget, time, and work progress, a project manager can only make decisions in the dark. Without a way to track errors and development progress, software development managers cannot make meaningful improvements in their processes. The more inadequate our metrics program, the closer we are to herding black cats in a dark room. The right metrics, used in the right way, are absolutely essential for project success.

Too many metrics are used simply because they have been used for years and people believe they might be useful. [2] Each metric should have a purpose, providing support to a specific decision making process. Metrics are too often dictated by leadership. They should be developed by a team under the direction of leadership. Metrics should be used not only by leadership but by all the various parts of an organization or development team. Obviously, all metrics that are useful to managers may not be useful to the accounting people or to developers. Metrics must be tailored to the users. The use of metrics should be defined by a program describing what metrics are needed, by whom, and how they are to be measured and calculated. The level of success or failure of your project will depend in a large part on your use or misuse of metrics – on how you plan, implement, and evaluate an overall metrics program.

While this chapter introduces and describes key metrics ideas and processes and can point you in the right direction, It is recommended you gain more insight, depth, and specific examples by downloading and reading those articles

and books listed in 8.5 Resources that apply to your effort. Of particular worth to DoD users is the *Practical Software and System Measurement Guidebook*.

8.2 Process Description

Metrics are not defined and used by themselves, but are part of an overall metrics program. This program should be based on the organization's goals and should be carefully planned, implemented, and regularly evaluated for effectiveness. The metrics program is used as a decision support tool. If the information provided through a particular metric is not needed for determining status or direction of the project, it's probably not needed at all. The role of the metrics program in the organization and its three major activities are shown in Figure 8-1.

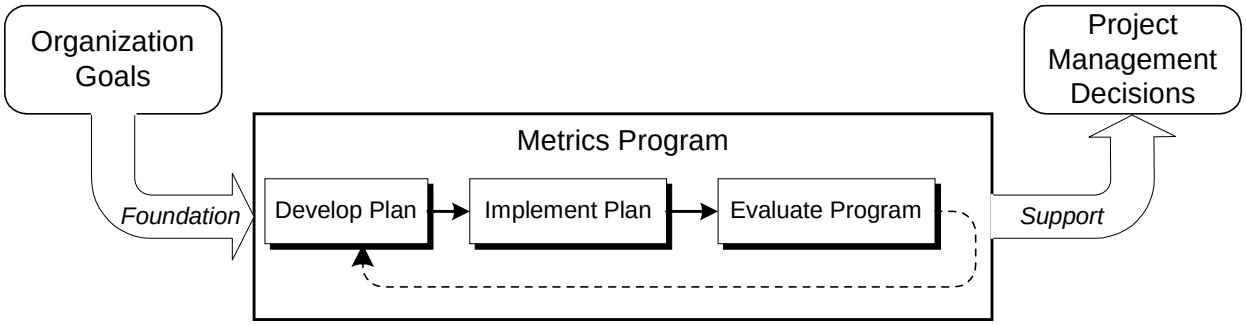

Figure 8-1 Metrics Program Cycle

8.2.1 Developing a Metrics Program Plan

The first activity in developing a metrics program is planning. Metrics planning is usually based on the goal-question-metric (GQM) paradigm developed by Victor Basili.

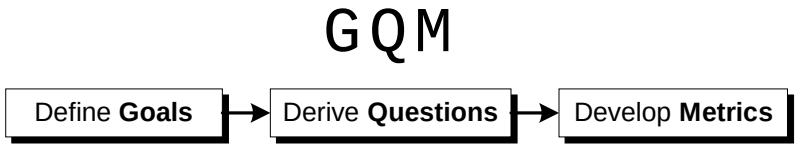

Figure 8-2 Basili's GQM Paradigm

The GQM paradigm is based on the following key concepts: [1]

1. Processes, including software development, program management, etc., have associated goals.
2. Each goal leads to one or more questions regarding the accomplishment of the goal.
3. Each question leads to one or more metrics needed to answer the question.
4. Each metric requires two or more measurements to produce the metric. (e.g. miles per hour, budget spent vs. budget planned, temperature vs. operating limits, actual vs. predicted execution time, etc.)
5. Measurements are selected to provide data that will accurately produce the metric.

The planning process is comprised of the three sub-activities implementing the GQM paradigm and one other, defining the data collection process. Each of these is discussed below.

8.2.1.1 Define Goals

Planning begins with well-defined, validated goals. Goals should be chosen and worded in such a way that they are verifiable, that is, their accomplishment can be measured or observed in some way. Goals such as meeting a specific delivery schedule are easily observable. Requirements stating "software shall be of high quality" are highly subjective and need further definition before they can be used as valid goals. You may have to refine or even derive your

own goals from loosely written project objectives. The selection or acceptance of project goals will determine how you manage your project and where you put your emphasis. Goals should meet the following criteria:

- They should support the successful accomplishment of the project's overall, or system-level, goals.
- They should be verifiable, or measurable in some way.
- They should be defined in enough detail that they are unambiguous.

8.2.1.2 Derive Questions

Each goal should evoke questions about how its accomplishment can be measured. For example, completing a project within a certain budget may evoke questions such as these: What is my total budget? How much of my budget is left? What is my current spending rate? Am I within the limits of my spending plan? Goals related to software time, size, quality, or reliability constraints will evoke different questions. It should be remembered that different levels and groups within the organization may require different information to measure the progress they are interested in. Questions should be carefully selected and refined to support the previously defined project goals. Questions should exhibit the following traits:

- Questions only elicit information that indicates progress toward or completion of a specific goal.
- Questions can be answered by providing specific information. (They are unambiguous.)
- Questions ask all the information needed to determine progress or completion of the goal.

Once questions have been derived which elicit only the complete set of information needed to determine progress, metrics must be developed which will provide that information.

8.2.1.3 Develop Metrics

Metrics are the information needed to answer the derived questions. Each question can be answered by one or more metrics. These metrics are defined and associated with their appropriate questions and goals. Each metric requires two or more measurements. Measurements are those data which must be collected and analyzed to produce the metric. Measurements are selected which will provide the necessary information with the least impact to the project workflow. Figure 8-3 summarizes the process of turning measurements into goal status.

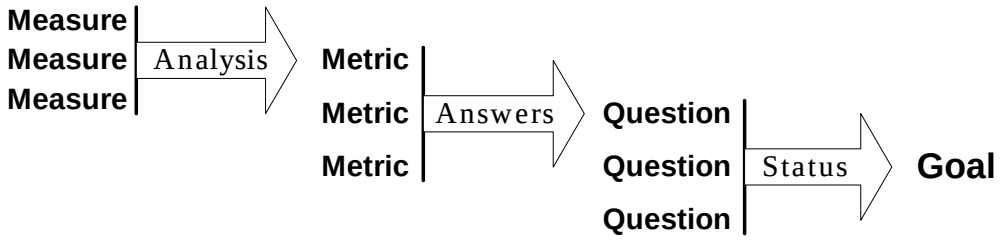

Figure 8-3 Flow of Measurements to Goal Status

Choosing measures is a critical and nontrivial step. Measurements that require too much effort or time can be counterproductive and should be avoided. Remember, just because something can be measured doesn't mean it should be. An in-depth introduction to measurements, *Goal-Driven Software Measurement–A Guidebook*, has been published by the Software Engineering Institute and is available as a free download at:

www.sei.cmu.edu/publications/documents/96.reports/96.hb.002.html

In addition to choosing what type of data is to be collected or measured, the methods of processing or analysis must also be defined in this step. How do you turn the measurements into a meaningful metric? How does the metric then answer the question? The analysis method should be carefully documented. Don't assume that it's obvious.

This activity is complete when you know exactly what type of data you are going to collect (what you are going to measure and in what units), how you are going to turn that data into metrics (analysis methods), and in what form (units, charts, colors, etc.) metrics will be delivered.

8.2.1.4 Define the Collection Process

The final step of the metrics planning process is to determine how the metrics will be collected. As a minimum, this part of the plan should include the following:

- What data is to be collected.
- What the source of the data will be.
- How it is to be measured.
- Who will perform the measurement.
- How frequently the data should be collected.
- Who the derived metrics will be delivered to, and in what format.

8.2.2 Implementing a Metrics Program

If the metrics program is well planned, implementing the program should be reduced to simply following the plan. There are four activities in the metrics implementation cycle, shown in Figure 8-4. [2]

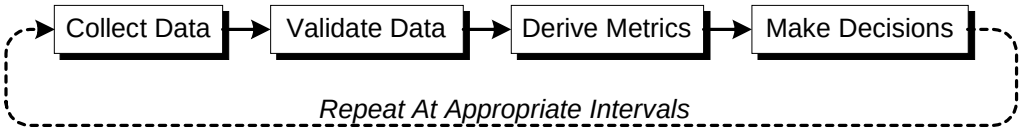

Figure 8-4 Metrics Implementation Cycle

Data is collected at specific intervals according to the plan. Data is then validated by examining the data to ensure it is the result of accurate measurements, and that the data collection is consistent among members of the group if it is being collected by more than one individual. In other words, is it being measured in the same way, at the same time, etc.? Once the data is determined to be valid, the metrics are derived by analyzing the data as documented in the metrics program plan. Metrics are then delivered to appropriate individuals and groups for evaluation and decision-making activities. This process is repeated until the project is complete.

8.2.3 Evaluating a Metrics Program

It is likely a metrics program will not be perfect in its first iteration. Soon after its initial implementation and at regular intervals after that, the metrics program should be evaluated to determine if it is meeting the needs of the metrics users and if its implementation is flowing smoothly. If metrics prove to be insufficient or superfluous, the program plan should be modified to provide the necessary information and remove any unneeded activity. The objective of a metrics program is to provide sufficient information to support project success while keeping the metrics program as simple and unobtrusive as possible. The following are areas that should be considered when reviewing a metrics program:

- Adequacy of current metrics
- Superfluity of any metrics or measures
- Interference of measurements with project work
- Accuracy of analysis results
- Data collection intervals
- Simplification of the metrics program
- Changes in project or organization goals

8.2.4 Metrics Repository

A final consideration is establishing a metrics repository, where metrics history is kept for future projects. The availability of past metrics data can be a goldmine of information for calibration, planning estimates, benchmarking, process improvement, calculating return on investment, etc. As a minimum, the repository should store the following:

- Description of projects and their objectives.
- Metrics used.
- Reasons for using the various metrics.
- Actual metrics collected over the life of the each project.
- Data indicating the effectiveness of the metrics used.

8.3 Measurement and Metrics Checklist

This checklist is provided to assist you in developing a metrics program, and defining and using metrics. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action. The checklist items are divided into three areas: developing, implementing, and reviewing a metrics program.

8.3.1 Developing a Metrics Program

- ❑ 1. Is your use of metrics based on a documented metrics program plan?
- ❑ 2. Are you using the GQM paradigm in developing your metrics?
- ❑ 3. Are your metrics based on measurable or verifiable project goals?
- ❑ 4. Do your goals support the overall system-level goals?
- ❑ 5. Are your goals well defined and unambiguous?
- ❑ 6. Does each question elicit only information that indicates progress toward or completion of a specific goal?
- ❑ 7. Can questions be answered by providing specific information? (Is it unambiguous?)
- ❑ 8. Do the questions ask for all the information needed to determine progress or completion of the goal?
- ❑ 9. Is each metric required for specific decision-making activities?
- ❑ 10. Is each metric derived from two or more measurements (e.g. Remaining budget vs. schedule)?
- ❑ 11. Have you documented the analysis methods used to calculate the metrics?
- ❑ 12. Have you defined those measures needed to provide the metrics?
- ❑ 13. Have you defined the collection process (i.e. what, how, who, when, how often, etc.)?

8.3.2 Metrics Program Implementation

- ❑ 14. Does your implementation follow the metrics program plan?
- ❑ 15. Is data collected the same way each time it is collected?
- ❑ 16. Are documented analysis methods followed when calculating metrics?
- ❑ 17. Are metrics delivered in a timely manner to those who need them?
- ❑ 18. Are metrics being used in the decision making process?

8.3.3 Metrics Program Evaluation

- ❑ 19. Are the metrics sufficient?
- ❑ 20. Are all metrics or measures required, that is, non-superfluous?

- ❑ 21. Are measurements allowing project work to continue without interference?
- ❑ 22. Does the analysis produce accurate results?
- ❑ 23. Is the data collection interval appropriate?
- ❑ 24. Is the metrics program as simple as it can be while remaining adequate?
- ❑ 25. Has the metrics program been modified to adequately accommodate any project or organizational goal changes?

8.4 References

[1] Perkins, Timothy K., "The Nine-Step Metrics Program," *Crosstalk* Magazine, February 2001: www.stsc.hill.af.mil/crosstalk/2001/feb/perkins.asp

[2] Augustine, Thomas, et al, "An Effective Metrics Process Model," *Crosstalk* Magazine, June 1999: www.stsc.hill.af.mil/crosstalk/1999/jun/augustine.asp

8.5 Resources

Army Software Metrics Office, Computer Tutorials (under "Products"): www.armysoftwaremetrics.org/

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "Metrics Tools: Effort and Schedule": www.stsc.hill.af.mil/crosstalk/1995/mar/metrics.asp
- "Project Recovery … It Can Be Done": www.stsc.hill.af.mil/crosstalk/2002/jan/lipke.asp
- "New Air Force Software Metrics Policy": www.stsc.hill.af.mil/crosstalk/1994/apr/xt94d04a.asp
- "Universal Metrics Tools": www.stsc.hill.af.mil/crosstalk/1995/sep/universa.asp
- "Software Metrics Capability Evaluation Methodology and Implementation": www.stsc.hill.af.mil/crosstalk/1996/jan/metrics.asp
- "Metrics Tools": www.stsc.hill.af.mil/crosstalk/1995/feb/metrics.asp
- "Really Bad Metrics Advice": www.stsc.hill.af.mil/crosstalk/1998/aug/backtalk.asp
- "A Methodic Approach to Effective Metrics": www.stsc.hill.af.mil/crosstalk/1994/oct/xt94d10c.asp
- "Why the New Metrics Policy": www.stsc.hill.af.mil/crosstalk/1994/apr/xt94d04b.asp
- "Metrics: Problem Solved?": www.stsc.hill.af.mil/crosstalk/1997/dec/metrics.asp
- "Metrics Tools: Size": www.stsc.hill.af.mil/crosstalk/1995/apr/metrics.asp
- "Quantitative Software Management Workshop - Software Cost and Schedule Estimation": www.stsc.hill.af.mil/crosstalk/1996/oct/xt96d10h.asp
- "Software Quality Metrics for Object-Oriented Environments": www.stsc.hill.af.mil/crosstalk/1997/apr/quality.asp
- "Metrics for Predicting Run-Time Failures and Maintenance Effort": www.stsc.hill.af.mil/crosstalk/1998/aug/predicting.asp
- "Metrics for Ada 95: Focus on Reliability and Maintainability": www.stsc.hill.af.mil/crosstalk/1995/may/ada95.asp
- "Pro-Active Metrics": www.stsc.hill.af.mil/crosstalk/1998/aug/proactive.asp
- "Best Measurement Tool Is Your Telephone": www.stsc.hill.af.mil/crosstalk/2001/mar/lucero.asp
- "Metrics Tools: Software Cost Estimation": www.stsc.hill.af.mil/crosstalk/1995/jun/metrics.asp
- "Software Maintainability Metrics Models in Practice": www.stsc.hill.af.mil/CrossTalk/1995/nov/Maintain.asp
- "Earned Value Project Management": www.stsc.hill.af.mil/crosstalk/1998/jul/value.asp

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, Chapter 13 & Appendix N, OO-ALC/TISE, May 2000. Download at: www.stsc.hill.af.mil/gsam/guid.asp

Literate Programming Software Metrics, many resources: www.literateprogramming.com/fmetrics.html

NASA Software Assurance Technology Center: http://satc.gsfc.nasa.gov/support/

- Recommended code metrics, by language: http://satc.gsfc.nasa.gov/metrics/codemetrics/index.html

- "Software Metrics and Reliability": http://satc.gsfc.nasa.gov/support/ISSRE_NOV98/software_metrics_and_reliability.html

Practical Software and Systems Measurement Guidebook, under "Downloads": www.psmsc.com

SEI Software Engineering Measurement and Analysis: www.sei.cmu.edu/sema/welcome.html

- "Software Measurement for DOD Systems: Recommendations for Initial Core Measures": www.sei.cmu.edu/publications/documents/92.reports/92.tr.019.html
- "Software Quality Measurement": www.sei.cmu.edu/publications/documents/92.reports/92.tr.022.html
- "Software Size Measurement": www.sei.cmu.edu/publications/documents/92.reports/92.tr.020.html
- "Goal-Driven Software Measurement–A Guidebook": www.sei.cmu.edu/publications/documents/96.reports/96.hb.002.html
- "Software Effort & Schedule Measurement": www.sei.cmu.edu/publications/documents/92.reports/92.tr.021.html
- "Software Cost and Schedule Estimating": www.sei.cmu.edu/publications/documents/94.reports/94.sr.003.html
- "Practical Software Measurement": www.sei.cmu.edu/publications/documents/97.reports/97hb003/97hb003abstract.html

This page intentionally left blank.

Chapter 9

Configuration Management

CONTENTS

This page intentionally left blank.

Chapter 9

Configuration Management

"But life is change, that is how it differs from the rocks, change is its very nature." – John Wyndham

9.1 Introduction

Change is a constant feature of software development. To eliminate change is to remove the opportunities to take advantage of lessons learned, incorporate advancing technology, and better accommodate a changing environment. Refusal to incorporate change can mean system limitations and early obsolescence, which in the world of technology can sign your system's death certificate before it is born. However, change is not universally benign and must be controlled in its introduction to a project.

9.1.1 Role of Change

All projects have as their main objective to change something. A system is being upgraded or replaced, presumably to provide better or greater functionality, ease of use, reduction of operating expenses, etc. As a project is executed, changes to the initial project plan and products are a natural occurrence. The following are common sources of changes:

- Requirements – the longer the delivery cycle, the more likely it is to happen.
- Changes in funding
- Technology advancements
- Solutions to problems
- Scheduling constraints
- Customer expectations
- Serendipitous (unexpected) opportunities for an improved system

Some of these changes may appear as options while others may be mandated from above or by circumstance, as in the loss of funding. In addition to these sources, many development efforts involve a progressive evolution or elaboration of capabilities and requirements. In projects of this nature the final product is expected to be the result of regular, controlled changes.

While all progress is accompanied by change, not all change is indicative of progress. If not properly handled, change can slip the schedule, affect the quality, and even kill the project. As a project draws closer to its completion, the impacts of change are more severe. [1] Clearly, a mechanism is needed to control change.

9.1.2 Configuration Management (CM)

In software development and other projects, proposed changes must be evaluated to determine their overall contribution to the project goals. Do they lead to improvements or do they ultimately impede or lower the quality of the project? Even those changes that are ultimately beneficial must be controlled in their introduction and implementation. Putting a bigger engine in a plane may improve its capabilities but it cannot be implemented until the aircraft structure has been found capable or been upgraded to support the increased weight and thrust.

Configuration Management is the process of controlling and documenting change to a developing system. It is part of the overall change management approach. As the size of an effort increases, so does the necessity of implementing effective CM. It allows large teams to work together in a stable environment, while still providing the flexibility required for creative work. [2] CM in a software environment is an absolute necessity.

CM has three major purposes: [1]

1. Identify the configuration of the product at various points on time.
2. Systematically control changes to the configuration.
3. Maintain the integrity and traceability of the configuration throughout the product life cycle.

CM accomplishes these purposes by answering and recording the answers to the change questions, who, what, when, and why, shown in Figure 9-1. [1] Being able to answer these questions is a sign of effective CM.

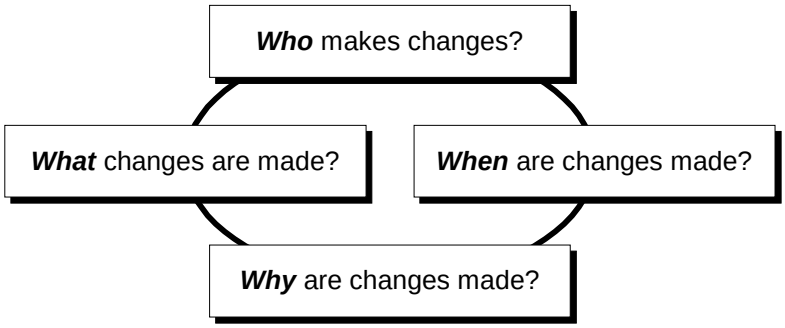

Figure 9-1 Configuration Management Questions

Effective CM provides the following essential benefits to a project:

1. Reduces confusion and establishes order.
2. Organizes the activities necessary to maintain product integrity.
3. Ensures correct product configurations.
4. Limits legal liability by providing a record of actions.
5. Reduces lifecycle costs.
6. Enables consistent conformance with requirements.
7. Provides a stable working environment.
8. Enhances compliance with standards.
9. Enhances status accounting.

In short, CM can provide cost effective project insurance when properly planned, organized, and implemented. [3] It must be integral to your overall project execution, and to your charter/customer agreement. [1] Proposed changes must be dealt with systematically, promptly, and honestly. [1] If the CM process is unreasonable or unresponsive, people will try to circumvent the process, leading to chaos and a loss of the benefits of true CM.

9.2 Process Description

While CM is a major element of a change control program, it is such a multifaceted discipline that it should be considered not simply as another activity, but as a program in and of itself. Establishing an effective CM program requires an understanding of CM functions and of the overall CM process.

The process presented herein is applicable to CM in general, including Software Configuration Management (SCM). If more details are needed for implementing SCM, please see documents specifically relating to SCM in Section 9.5, Resources.

9.2.1 Functions of Configuration Management

Configuration Management is comprised of four primary functions, Identification, Control, Status Accounting, and Auditing. These are shown in Figure 9-2, along with their sub-functions. All CM activity falls within the bounds of these functions.

9.2.1.1 Identification

This function identifies those items whose configuration needs to be controlled, usually consisting of hardware, software, and documentation. These items would probably include such things as specifications, designs, data, documents, drawings, software code and executables, components of the software engineering environment (compilers, linkers, loaders, hardware environment, etc.), and hardware components and assemblies. Project plans and guiding documents should also be included, especially the project requirements. A schema of names and numbers is developed for accurately identifying products and their configuration or version level. This must be done in accordance with project identification requirements. Finally, a baseline configuration is established for all configuration items and systems. Any changes to the baselines must be with the concurrence of the configuration control organization. [2]

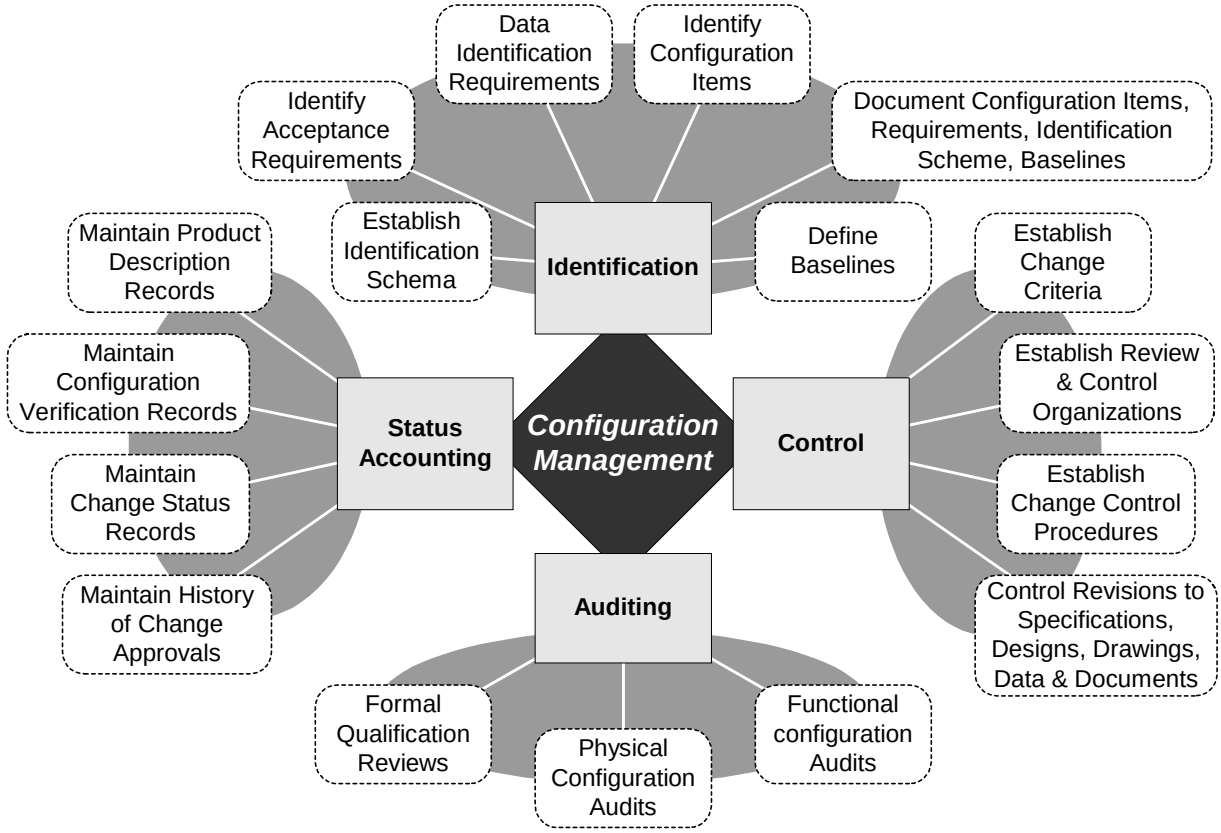

Figure 9-2 Major Functions of Configuration Management [2]

9.2.1.2 Control

Configuration control establishes procedures for proposing or requesting changes, evaluating those changes for desirability, obtaining authorization for changes, publishing and tracking changes, and implementing changes. This function also identifies those persons and organizations that have authority to make changes at various levels (configuration item, assembly, system, project, etc.,) and those who make up the configuration control board(s). Additionally, various change criteria are defined as guidelines for the control organizations. Different types of configuration items or different systems will probably need different control procedures and involve different people. For example, software configuration control has different needs and involves different people than communications configuration control and would probably require different control rules and a different control board. [2]

9.2.1.3 Status Accounting

Status accounting is the documentation function of CM. Its primary purpose is to maintain formal records of established configurations and make regular reports of configuration status. These records should accurately describe the product, and are used to verify the configuration of the system for testing, delivery, and other activities. Status ac-

counting also maintains a history of change requests and authorizations, along with status of all approved changes. This includes the answers to the CM questions in Figure 9-1. [2]

9.2.1.4 Auditing

Effective CM requires regular evaluation of the configuration. This is done through the auditing function, where the physical and functional configurations are compared to the documented configuration. The purpose of auditing is to maintain the integrity of the baseline and release configurations for all controlled products. [2] Auditing is accomplished via both informal monitoring and formal reviews.

9.2.2 Configuration Management Process

Understanding of what CM is supposed to accomplish is one thing. Putting it into practice is another. As with most project activities CM begins with planning. With a plan, configuration baselines can be established. Following this initial configuration identification, the cyclical configuration control process is put into motion. These three major CM implementation activities are shown in Figure 9-3.

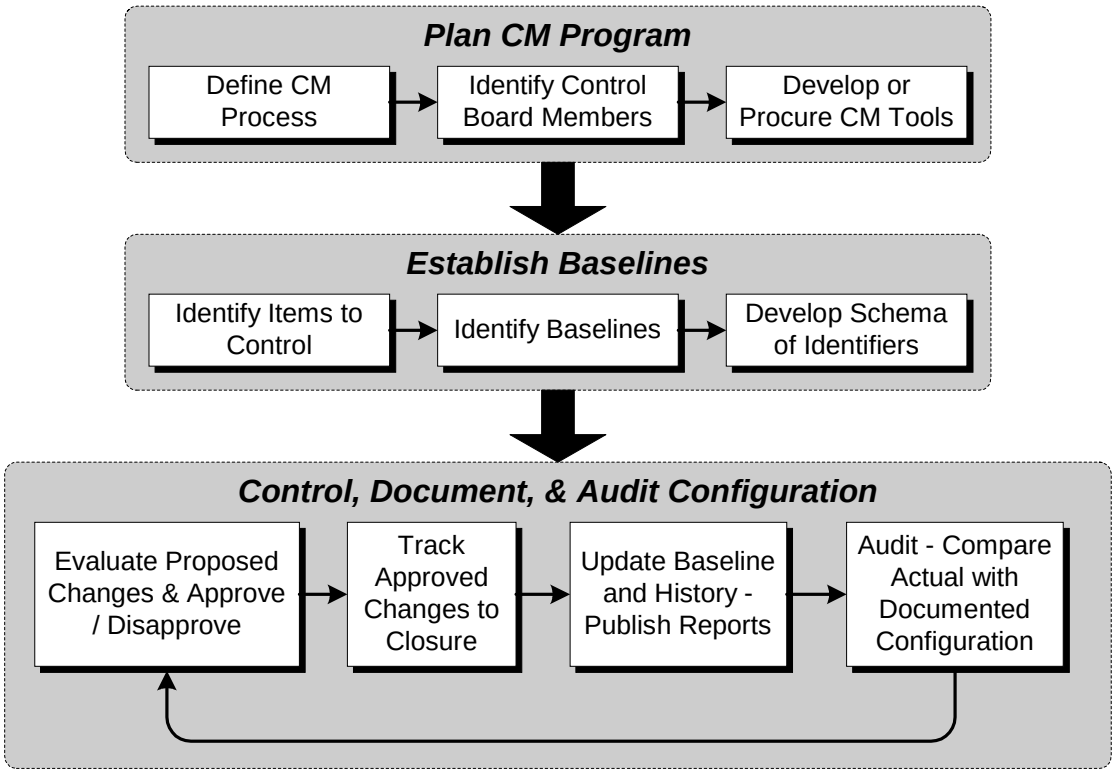

Figure 9-3 Configuration Management Implementation Process

9.2.2.1 Planning

Planning begins by defining the CM process and establishing procedures for controlling and documenting change. A crucial action is the designation of members of the Configuration Control Board (CCB). Members should be chosen who are directly or indirectly involved or affected by changes in configuration. For example, a software configuration change board would obviously be populated with representatives from different software teams, but software affects many more aspects of a project. There should also be representatives from the hardware, test, systems, security, and quality groups, as well as representatives from project management and possible other organizations. Not all changes would be reviewed by this august body. Changes occur at different system levels and affect different portions of the overall system. Many changes will probably only affect a small subset of the system and could therefore be reviewed and approved by a smaller group. Some sort of delineation of change levels should be made during planning to keep change decisions at the proper level. While software CM is essential, there may need to be other

CCBs to control change in other areas of the project. Again, this is something that should be worked out in the planning phase.

Various software tools exist which can facilitate the CM process flow and maintain configuration history. Use of a CM software tool is highly recommended. The temptation will be to choose a tool because it looks good in a demonstration and then build the CM process around it. The process should be defined first, and then a tool chosen to facilitate the process.

9.2.2.2 Establishing Baselines

Once the CM program exists on paper, it must be determined just what configurations it will control. The second major step of implementing effective CM is identifying what items, assemblies, code, data, documents, systems, etc. will fall under configuration control. With the configuration items identified, the baseline configuration must be identified for each item. For items that already exist it may prove to be nothing more than examining or reviewing, and then documenting. For those items that have not been developed yet, their configuration exists in the requirements database or in the project plans. Until they come into physical or software reality, changes to their configuration will consist only of changes to the requirements or plans.

Another essential activity in this step is developing a schema of numbers, letters, words, etc. to accurately describe the configuration revision, or version, for each general type of configuration item. There may be project requirements which dictate some type of nomenclature, or there may be an organizational or industry standard that can be used as the basis for configuration identification.

9.2.2.3 Controlling, Documenting, and Auditing

When the baselines have been established, the challenge becomes one of keeping the actual and documented configurations identical. Additionally, these baselines must conform to the configuration specified in the project requirements. This is an iterative process consisting of the four steps shown in Figure 9-3. All changes to the configuration are reviewed and evaluated by the appropriate configuration control representatives specified in the CM plan. The change is either approved or disapproved. Both approvals and disapprovals are documented in the CM history. Approved changes are published and tracked or monitored until they are implemented. The appropriate configuration baseline is then updated, along with all other applicable documents, and reports are published and sent to affected organizations indicating the changes that have occurred. At selected time intervals and whenever there appears to be a need, products and records are audited to ensure that:

- The actual configuration matches the documented configuration.
- The configuration is in conformance with project requirements.
- Records of all change activity are complete and up-to-date.

The controlling, documenting, auditing cycle is repeated throughout the project until its completion.

9.2.3 Updating the CM Process

It is unlikely a perfect CM program will be assembled during the initial planning stage. There will be learning and changes in the program that indicate a need for adjustments in the CM process. These may be any mixture of modifications to make it more efficient, responsive, or accurate. When changes in the CM process are needed, consider them as you would any other changes, get the approval of all participating organizations, and implement them as appropriate. It would be ironic indeed to have an unchanging change process.

9.3 Configuration Management Checklist

This checklist is provided to assist you in establishing an effective CM program. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

9.3.1 CM Planning

- ❑ 1. Have you planned and documented a configuration management process?
- ❑ 2. Have you identified Configuration Control Board members for each needed control board?

- ❑ 3. Has CM software been chosen to facilitate your CM process?

9.3.2 Establishing Baselines

- ❑ 4. Have all configuration items been identified?
- ❑ 5. Have baselines been established for all configuration items?
- ❑ 6. Has a descriptive schema been developed to accurately identify configuration items and changes to their configuration?

9.3.3 Controlling, Documenting, Auditing

- ❑ 7. Is there a formal process for documenting and submitting proposed changes?
- ❑ 8. Is the Configuration Control Board active and responsible in evaluating and approving changes?
- ❑ 9. Is there a "higher authority" to appeal to when the CCB gets "hung," and can't come to a consensus?
- ❑ 10. Are all changes tracked until they are fully implemented?
- ❑ 11. Are all changes fully documented in the baseline documents and change histories?
- ❑ 12. Are regular reports and configuration updates published and distributed to interested organizations?
- ❑ 13. Are regular audits and reviews performed to evaluate configuration integrity?
- ❑ 14. Are configuration errors dealt with in an efficient and timely manner?

9.3.4 Updating the Process

- ❑ 15. Is the CM program itself – its efficiency, responsiveness, and accuracy – evaluated regularly?
- ❑ 16. Is the CM program modified to include recommended improvements when needed?

9.4 References

[1] Software Technology Support Center Course: *Life Cycle Software Project Management*, Configuration Management, 9 October 2001.

[2] *Little Book of Configuration Management*, Software Program Managers Network, November 1998. Download at: www.spmn.com/products_guidebooks.html

[3] *Program Manager's Guide for Managing Software*, 0.6, 29 June 2001, Chapter 14: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

9.5 Resources

CM Today, Configuration management papers: www.cmtoday.com/yp/papers.html

Configuration Management Resource Guide: www.quality.org/config/CMResourceGuideMaster.doc

Configuration Management II Users Group: www.cmiiug.com

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "Software Configuration Management Terminology": www.stsc.hill.af.mil/crosstalk/1995/jan/terms.asp
- "Software Configuration Management: Function or Discipline?": www.stsc.hill.af.mil/crosstalk/1995/oct/cmfunct.asp
- "Stop-Gap Configuration Management": www.stsc.hill.af.mil/crosstalk/1998/feb/stopgapcm.asp
- "Effective Software Configuration Management": www.stsc.hill.af.mil/crosstalk/1998/feb/effectivecm.asp
- "Configuration Management Web Sites": www.stsc.hill.af.mil/crosstalk/1999/mar/cmsites.asp
- "Demystifying Software Configuration Management": www.stsc.hill.af.mil/crosstalk/1995/may/demystif.asp

- "Evaluation and Selection of Automated Configuration Management Tools": www.stsc.hill.af.mil/crosstalk/1995/nov/evaluati.asp
- "Software Configuration Management: A Discipline with Added Value": www.stsc.hill.af.mil/crosstalk/2001/jul/butler.asp
- "Introducing Process into Configuration Management": www.stsc.hill.af.mil/crosstalk/1996/jun/introduc.asp
- "Process-Based Configuration Management": www.stsc.hill.af.mil/crosstalk/1997/apr/configuration.asp
- "Achieving the Best Possible Configuration Management Solution": www.stsc.hill.af.mil/crosstalk/1996/sep/achievin.asp
- "A Tutorial on Control Boards": www.stsc.hill.af.mil/crosstalk/1999/mar/sorensen.asp
- "Adopting SCM Technology": www.stsc.hill.af.mil/crosstalk/1996/mar/adopting.asp

Department of Energy, Configuration Management resources: http://cio.doe.gov/sqse/pm_conf.htm

Institute of Configuration Management: www.icmhq.com

MIL-HDBK-61A, Configuration Management Guidance: www.assist2.daps.dla.mil/quicksearch/

Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapter 14: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Software Engineering Institute, Software Configuration Management support: www.sei.cmu.edu/legacy/scm/scmHomePage.html

This page intentionally left blank.

Chapter 10

Software Engineering Processes

CONTENTS

This page intentionally left blank.

Chapter 10

Software Engineering Processes

"Because software, like all capital, is embodied knowledge, and because that knowledge is initially dispersed, tacit, latent, and incomplete in large measure, software is a social learning process ... in which knowledge that must become the software is brought together and embodied in the software." – Howard Baetjer, "Software as Capital"

10.1 Introduction

In the earliest days of computers, memory was small, language consisted of binary machine code, and programmers were adventurous souls from other disciplines who made up the craft of programming. While early programmers could get by writing some code and using clever tricks, often undocumented, those were not the "good old days." As technology advanced, so did the need to build bigger and ever more complex programs. Software development reached a point where it had to be developed by teams. There had to be requirements, plans, detailed designs, actual building, testing, development of efficient, intuitive user interfaces, and integration into larger systems of computers, machines, and people. The discipline to develop software in this manner is known as software engineering, a complex process that itself requires many sub-processes.

To put up a tent you need a level spot of ground, located in a small clearing. A tent is a relatively simple form of shelter with few requirements. It can be erected quickly, with little planning, and at little cost, but it does have its limitations. Contrast this with building a house. Much more is needed by way of planning, materials, cost, time, and work. Presumably, the builder or future owner of the house has some specific and general requirements to be met. These will need to be turned into some type of floor plan, which must then be approved by the requirement giver. There may even be artists' renditions of what the completed house will look like. Following the top-level plan, designers will put together detailed plans for each room and all the major components of the house. Again, these must be approved. When the design is satisfactory, the building will start and proceed until the house is finished. There will be inspections along the way and inspectors will examine and test various parts of the house for quality, functionality, and adherence to the plans and standards. There will probably be some changes during the actual building to make up for unknown variations in the building lot, available materials, or other factors. The future owner may even request a change after seeing what the plan is going to produce. When the structure is complete, designers or owners will decide on paint colors, carpets, lighting, and furniture. Sometime before being used, the house will need to be connected to various utilities, sidewalks, and roads. This will make it a part of the community. The primary differences between tent and house are the levels of complexity and functionality. Requirements for greater functionality result in a more complex system, which requires a more complex construction process.

Software engineering is younger than other engineering disciplines and is based on an industry that is undergoing unbelievable change. Because of its relative youth, the software industry tends to be populated by younger, creative, innovative people. All these combine to produce a constantly evolving discipline. Just as software life cycles have multiplied (see Chapter 2, Software Life Cycle.), so too has software engineering diverged into many different methods, each with its own disciples and champions. It is not possible to sample even the major methods here, but most development methods incorporate, to a greater or lesser degree, a standard set of activities common to most software engineering. These common pieces, framed in a "standard" engineering methodology will form the basis of our study.

10.2 Process Description

Software engineering goes far beyond learning how to write programs with computer languages. While there are aspects of software engineering which are used more effectively, or even exclusively, with specific languages, the discipline is not dependent upon language, notwithstanding the divine qualities some programmers may ascribe to their chosen language of worship. The following statement warns of the true position of programming in the scheme of software engineering; there is much more to software development than programming.

"The sooner you begin writing code, the longer it will take to get done."

Much of software engineering is associated with the software life cycle process employed (See Chapter 2, Software Life Cycle). The software development model used in a project is a portion of the overall project life cycle, and should fit within the project life cycle. The classic software development waterfall model is shown in Figure 10-1. [1] This model embodies all the major aspects of the software engineering process. The prototyping activity is not part of the classic model but is commonly used to provide feedback and allow iterative development where requirements or solutions need further definition. Other development models are often used. In addition to Figures 10-2 and 10-3, the life cycle models presented in Chapter 2 show some of the more common models.

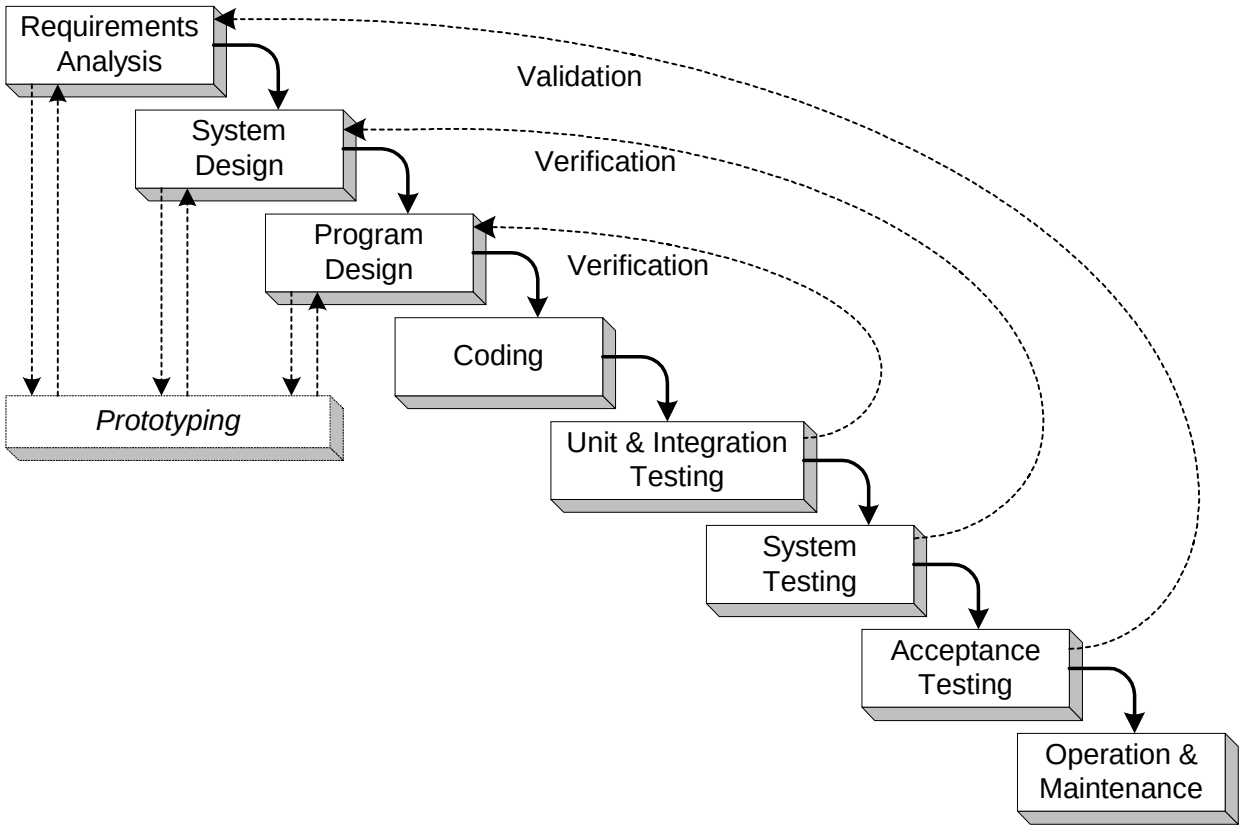

Figure 10-1 Classic Waterfall Software Development With Prototyping Modification [1]

The waterfall model has each activity flowing into the subsequent activity following its completion. While engineers look ahead and try to anticipate issues and needs that will surface later in the project, the waterfall is limited in its ability to provide feedback and allow the work of earlier steps to be modified. The prototyping modification shown in Figure 10-1 can help alleviate this deficiency somewhat.

The V model improves on the sequential nature of the waterfall model by planning and preparing acceptance tests in conjunction with the requirements analysis. System tests are established during system design and unit and integration tests are planned during program design. The tests are not performed until later in the development process, but developing tests in conjunction with the applicable requirement or design activity facilitates a more unified and focused development effort. Figure 10-2 depicts the V model.

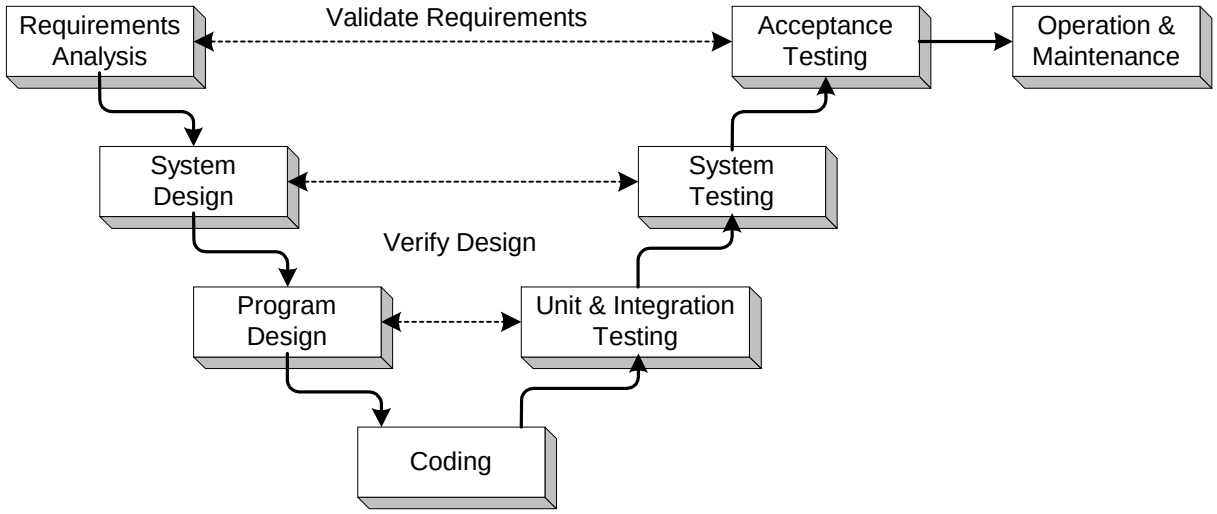

Figure 10-2 V Software Development Model [1]

The shark tooth model, shown in Figure 10-3, incorporates prototyping and involves the user in providing feedback to the development process. Users are involved in the system requirements analysis, following which, the developers perform software requirements analysis. A prototype is produced and shown to the user. Requirements are refined and development proceeds through different design phases, involving reviews and further prototype demonstrations as necessary. The prototypes allow the user and developer to better understand each other and produce a product more in line with the user's real needs.

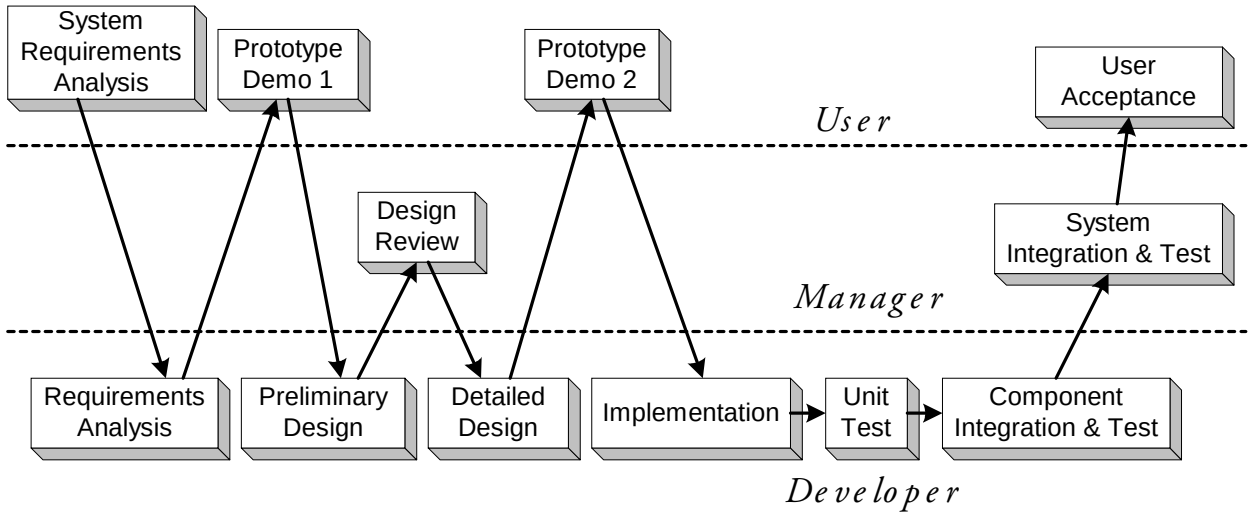

Figure 10-3 Shark Tooth Development Model [1]

The DoD requires the use of the spiral development model (see 2.2.1.4) wherever possible. This model is more iterative than the shark tooth model and at first glance appears far more complex. However, careful inspection will reveal that it employs the same software engineering activities as the other models. In other words, there are several development models, along with variations on those models, but there is a standard set of software engineering principles which can be found within the framework of most development life cycles. It is the job of the Software Engineer (SE) to apply these principles in a systematic, disciplined, and quantifiable approach to software development. [1] A partial list of the skills and knowledge areas that make up software engineering is shown below.

- Project Planning
- Requirements & Specification
- Software Architecture

- Design Methods
- Data Structures
- Algorithms
- Modeling
- Testing
- Demonstrations
- Software Components Design
- Languages, levels, constructs, syntax
- Human Factors
- Measurement & Estimation
- Verification and Validation
- Communications
- Coding (Programming)
- Operating Systems
- CASE Tools
- Software Quality
- Numerical methods
- Life Cycles

Key areas of this list are discussed in the following paragraphs.

10.2.1 Software Project Planning

As in every other discipline, planning is an essential skill for developing software. Planning includes determining what activities need to be done, in what order, when, and by whom. It also includes the following:

- What tools will be used
- How activities will be divided
- Scheduling
- Requirements management
- Quality management
- Design methods to be used
- Problem solving approach
- Demonstration methods
- Documentation, internal & deliverable
- Test methods
- Configuration control
- Design reviews
- Risk management

Responsibility for all this planning is shared by the project manager and members of the development team, including individual engineers. Each member of the team must be able to plan, or contribute to the planning of those areas with which he or she is involved.

10.2.2 Requirements and Specification

Requirements form the basis of the goals for which software is developed. Seldom are requirements known in complete detail, and engineers must elicit requirements from stakeholders to find out what is really needed and wanted. It is far more than asking questions. It involves developing and asking the right questions, perhaps educating both developers and users so they can reach a point of common understanding. It also consists of being able to communicate with users in such a way as to make sure each party knows how the other perceives the requirements.

The requirements process, summarized in Figure 10-4, has two major activities and two major products. The requirements gathered during the elicitation activity are documented in a system specification. This document is written in natural language (common everyday speech) and describes what the final system or software should do for the users. It can be general in nature or very detailed. Both stakeholders and developers should be in agreement that the system specification adequately and accurately describes the new system and its capabilities. Note that some analysis is usually required to produce the system specification.

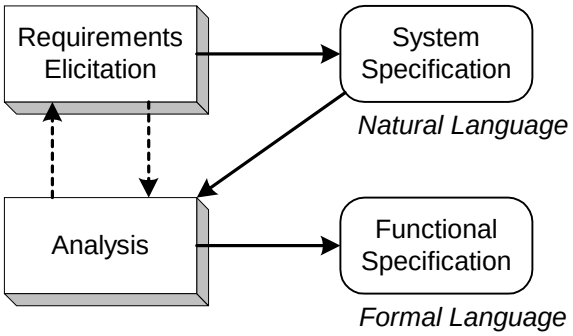

Figure 10-4 Requirements Elicitation and Analysis [1]

If it is not possible to implement the system requirements due to time, money, technology, or other constraints, the developers negotiate with the stakeholders to agree on a set of requirements that can be performed under the given constraints.

When the system specification is complete, its requirements are analyzed to determine what will be needed to comply with them. Analysis is the primary activity here, with occasional returns to the users to elicit information to clar-

ify ambiguities. The product of the requirements analysis is a functional specification containing detailed requirements. The functional specification not only differs in its level of detail from the system specification, but also in its format. It is a formalized description of the system, presented in a language that better describes what must be accomplished in terms of software functionality. It gives specifics of what must be accomplished. It often includes relationship diagrams, data flows, use case diagrams, and logic flows. To those uninitiated into the world of formalized software specification, it might appear as a foreign language using foreign mathematics. But to the software engineering world it is the exact bridge needed to cross from the systems specification to the design process.

10.2.3 Design

Design is generally performed at two primary levels, the overall system design and the design of lower level components, also known as program design. While these two levels are summarized here, it should be remembered that there may be a prototype building cycle in the development plan. In that case, there would be one or more iterations of design-build-test to get a better idea where the system is going, before the final lower design is performed.

10.2.3.1 Systems Level Design

The first design activity is to develop a system level concept of what the software will look like, accomplish, and how it will participate in the overall system of hardware, software, and humans. This includes defining interfaces with the world external to the system, including the user interfaces, the various software subsystems, along with their functions, and the interfaces and data flows between them. The system and software specifications are used extensively to ensure the system is being built to satisfy the requirements. The systems design defines what goes into and out of the system, what functions are performed, and by what subsystems. The design is generally presented for approval or modification at a preliminary or system design review.

10.2.3.2 Program Design

Program design takes the overall structure, functionality, interfaces, and data flows of the system and adds detail. The structure is defined down to the individual program modules. This is done by dividing design into smaller steps, such as *Top Level Design* and *Detailed Design*. Subdividing the design process implements the divide and conquer principle, making design easier and more controlled while facilitating the detection of errors.

All necessary processing and sub-functions that make up the system functions are named, defined, and allocated to specific modules. Because different people may be working on different modules, interfaces must be defined exactly. Data flows and data storage are mapped out and defined in detail. Algorithms, mathematical and other processing are all defined. Individual modules are designed using pseudo-code or natural language to describe what each module does. Data and variables used to interface with other modules are precisely defined.

The completed design is usually presented for formal approval at a critical design review. This is critical as the name says. It is used to spot any inconsistencies, logic errors, interface mismatches, and forgotten functions.

The temptation will always exist to leave more of the design to the coding activity. Remember, "*the sooner you begin writing code, the longer it will take to get done."* The more complete and detailed the design is, the faster and easier coding can be accomplished. Better design also means fewer errors, easier debugging, and less time spent in the code and test iterations. Good, well-documented designs also make software maintenance much easier and far less costly than it would otherwise be.

10.2.3.3 CASE Tools

Computer Aided Software Engineering (CASE) tools are software programs that assist the software engineer in designing and documenting software. They are usually dependent on the design methodology employed by the development team, and may even be dependent on the language chosen for implementing the design. Their chief contributions consist of the following advantages:

- Encourage or enforce a formal design process or methodology.
- Document the design in a consistent, formal manner.
- Track the details to help ensure things aren't forgotten.
- Unify the development team in their efforts.

In addition to the above advantages, some CASE tools help in discovering errors, developing tests, tracking requirements, and other benefits. Every effort should be made to select an easy-to-use, functional, helpful, and proven tool that not only integrates with your development process, but actually facilitates, assists, and documents that process in as many areas as possible.

10.2.4 Coding with Programming Languages

Coding, or programming, as we have discussed, is only a part of the software engineering process. For most engineers, it is the favorite part, where you put things into the computer and it responds. This is probably the greatest source of temptation to start coding right away, and to try to wrap the other aspects of software engineering into the coding process. Sadly, the desire to move prematurely into programming, to "save time" or "save money," even extends to some leaders and managers, who may be under pressure by those above them who know little of the development process. There is a joke where the manager says to the programmers, "I'll go up and find out what they need and the rest of you start coding."

Programming is where planning, specifying, and designing take on real existence, at least as much as a non-tangible entity can. This is where the engineer creates actual programs that execute on a computer, receiving inputs and providing outputs. Programming starts with coding individual modules or functions. These units are then integrated together in progressively larger and more complex subsystems, until the entire software system is brought together and executed as a system. Each step of integration shows the software units can work together or discovers problems with their interaction. Programming involves these major knowledge areas:

- The programming language
- General programming skills
- The Operating System (OS)
- Hardware that may be associated with the software

10.2.4.1 Programming Languages

There are countless programming languages. They have been developed or created since the beginning of the computer era. Some are general purpose and have been converted to run on many different platforms or computer systems. Others are narrow in their application and have been made for specific purposes or for specific computers. They are called by many names: Cobol, Fortran, C, Basic, APL, Pascal, Ada, C++, Java, FoxPro, C#, Assembly, etc. Each language has specific reserved words it uses to command the computer to perform certain operations. These reserved words must be used in a certain syntax, the order or structure of programming statements, for the computer understand them. In addition, each language has requirements for the structure of the overall program, how it implements general programming operations, how it deals with the computer, and how a program is written, compiled, and executed. Most programmers know more than one language but have a favorite. Learning to use a programming language, like spoken languages, takes training (formal or informal) and practice.

A computer language should be selected for use only after a careful review of the systems and software engineering factors that influence the overall life cycle costs, risks and potential interoperability. [2] When choosing a language, all factors considered should be documented along with the decision process and the results. Major factors to consider are:

- Reliability
- Safety
- Standards
- Development Methodologies
- Cost
- Performance
- Security
- Development Tools
- Maintenance & Supportability
- Schedule
- Interoperability
- System Architecture
- Software Architecture
- Staffing
- Open Systems

10.2.4.2 General programming skills

General programming skills exist apart from specific languages and may be implemented by most languages, albeit in different manners. They include operations (loops, branches, Boolean logic, etc.), structures (arrays, strings, objects, etc.), algorithms (sorting, mathematical, parsing, compression, etc.), data types (strings, integers, floating point, Boolean, etc.), input/output, threads, inheritance, etc. These are learned to some extent by studying theory and to a greater extent by actual programming. Because skills are usually learned to satisfy a need – "How do I sort this efficiently?" – rather than curling up with a good algorithm book in front of the fireplace, experience can play a big part in producing good, efficient program code. This is the area where skilled programmers practice their "art" and employ their tricks.

Another subject belonging to skills is the use of programming style. Just as written language is easier to understand and use when certain style conventions are used, programs can be made easier to follow, understand, debug, test, and maintain if certain style conventions are employed. Generally, this will be one of the project requirements. Various style guides exist for languages in the industry and within many organizations.

10.2.4.3 Operating Systems

The operating system (OS) is a program that acts as the heart and soul of the computer. When a computer is powered up the OS runs through all the system checks and sets up memory and other resources for use by other programs. The OS receives instructions from and presents information to the computer operator and launches other programs as directed. As in programming languages, people prefer one OS over another and often ascribe personality traits to the various OS's they know or have heard of. Unlike languages, which are often associated with divine attributes, OS's are more often than not ascribed diabolical qualities, and viewed as inherently evil entities that must be bypassed or appeased to get anything done.

Software engineers must come to know the OS that resides on the computer for which they are writing software. The OS will in most cases be running at the same time as the new software. In spite of having to figure out ways to work around the OS to accomplish certain software feats, in many cases the OS will perform various functions outside of the new program. For example, why write a driver and interface program for a keyboard, monitor, or printer when one can simply ask the OS to perform that function? Knowing how the OS operates and what it can and cannot do for you is essential for programming.

10.2.4.4 Hardware Systems

Because software cannot be touched, it doesn't exist outside a computer system. Software is meaningless without hardware to exist in. Each type of computer system has its own capabilities, resources, and idiosyncrasies. This is true for all systems, whether they are general-purpose desktop computers or embedded computers in a microwave oven or weapon system. Memory, storage, communications, calculation speed, and interfaces are all constrained and defined by actual physical hardware. To use them, the software engineer must have an understanding of what hardware is available, how it is used or accessed, what its limitations are, and how it is controlled by software. By itself, hardware is just silicon and metal. Software is intangible. Only together do they form a functional system. One cannot be understood without the other.

10.2.5 Testing, Validation, and Verification

Software must be evaluated at intervals during development and when complete to determine if it actually performs as it should. Testing is done at various stages from the coding phase throughout the remainder of the project. Individual programmers test individual modules or units of the program as they are programming. This helps them spot errors in both logic and coding. It also shows that the module is correct and functional. During the integration process, modules are tested together in subsystems to detect errors with module interfacing and with working together. Additionally, integration testing determines whether the larger groupings of modules function smoothly and correctly together. Testing continues with larger and larger groups until the full system is evaluated. If testing is done correctly throughout the development process, the final acceptance testing should be a demonstration of the full, correct functionality of software and its fulfillment of requirements. There should not be any surprise errors or nonperformance. The hierarchy of testing is shown in Figure 10-5.

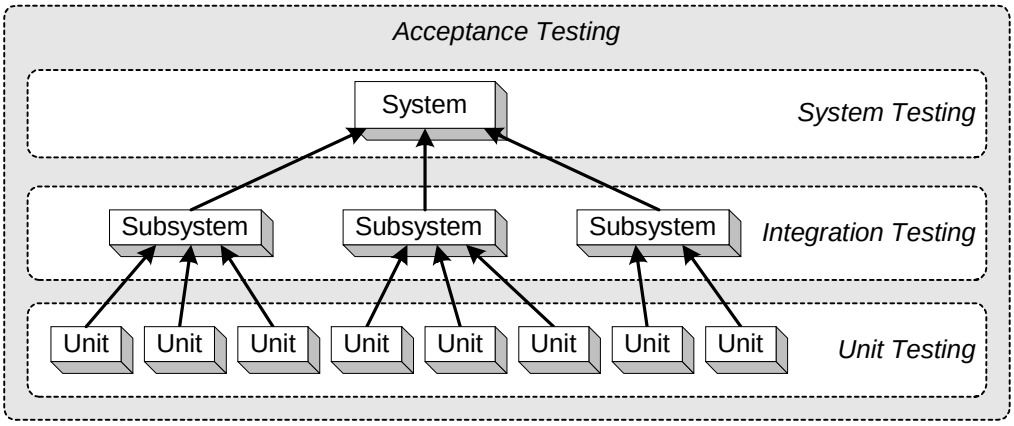

Figure 10-5 Testing Hierarchy

When evaluating software, two specific types of performance are considered: validation and verification. Validation is the determination of whether the right software has been built. Does it meet the requirements established in the beginning phases of the project? Verification is determining whether the software has been built according to the design. A system can be built right without building the right system. In other words, the software may be verified and not validated. Ultimately, the system must meet both criteria, being built according to the design and the requirements, if it is to be useful.

Figures 10-1 and 10-2 show where verification and validation testing are performed and what drives them. It should be noted that although acceptance testing validates the system for the stakeholders, acceptance testing should be a formality. The real testing should have been performed throughout unit testing, integration, and system testing.

10.2.6 Human Factors

While many automated systems deal only with machines, most have at least some interaction with humans. Software engineering includes understanding and taking into account the human aspect of the man-machine interface. This includes human sensory, psychological, and bio-mechanical considerations. Some software functions almost exclusively as an interface to humans. Other software requires human interaction only for startup and occasional direction. Software functionality is worth very little if the user cannot easily and efficiently control the software. Great care and consideration must be given to interaction between the computer and the user. This includes such things as intuitiveness, color, light, noise, redundancy, input methods, and display/output methods of the interface, as well as fatigue, stress, and boredom of the user. Developers must also understand information theory and human information processing. While failure to incorporate human factors into the development process may result in poor sales for a computer game, in a logistics system it can lead to people not using the software tool and bypassing the system. In an avionics or weapons system it can lead to injury or death.

10.2.7 Maintenance

When software is fielded or released to the users there will be a period of training and learning. If the users find things they would like to add or change, and that is likely, the stage is set for upgrades to the software. While we have talked extensively about software development, most of the money spent on software, 70% or more, is spent on maintenance. Maintenance may consist of fixing a problem not discovered in the development phase, adding new functionality to the software, or modifying the software to deal with changes in the rest of the system. While maintenance is usually the longest phase of the software life cycle, it is dealt with in a manner similar to previous development efforts.

Software maintenance projects are development projects that begin with previously constructed software. Using that software becomes one of the requirements of the new, improved system. The maintenance project, like other projects, consists of planning, requirements, design, coding, and testing. Software maintenance will generally go on in a series of discrete upgrade projects as needs arise and as resources are available until the overall system is retired and/or replaced. The maintenance cycle is shown in Figure 10-6.

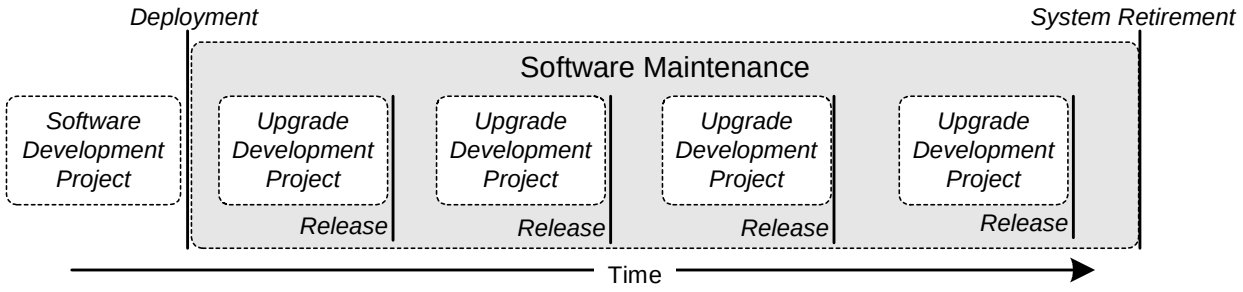

Figure 10-6 Maintenance Upgrade Cycle

10.2.8 Summary

This chapter has provided only a brief overview of primary software engineering activities or knowledge areas. Many other areas, such as those in the list in Section 10.2, have been left out because of time and space constraints, not because they are not essential. Better understanding of the software engineering process can be gained by studying those areas in the reference material listed at the end of this chapter.

Those who become software engineers go through a rigorous course of study, training, and practice to gain knowledge, develop skills, and gain insight into the software development process. The discipline is the application of engineering principles to software development and brings a number of knowledge fields together in a coordinated effort. Because of the vast amount of man years spent developing software there have been many innovations and improvements to the engineering process which have not only benefited the software industry but have migrated into other engineering disciplines as well. Because of the constant improvement of the computer industry and the incorporation of computers into more and more tools, toys, and other systems, keeping up with the advances in software engineering is a full time, and probably lifetime, process.

10.3 Software Engineering Processes Checklist

This checklist is provided to assist you in understanding the software engineering issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

10.3.1 Before Starting

- ❑ 1. Do you know what software development life cycle your project will be employing and how it coordinates with the software and project life cycles?
- ❑ 2. Does the development team have experience in the software development life cycle to be used?
- ❑ 3. Do the developers, the stakeholders, and you understand what the steps of the development process are, and what the inputs and products of each step are?
- ❑ 4. Has your project been planned in the various software development areas listed in Section 10.2.1 and in Chapter 3?
- ❑ 5. Do you know what design method has been chosen for the development effort and why it was chosen over other methods?
- ❑ 6. Does your development team have experience with the chosen design method? If not, has the schedule been adjusted to allow for learning the new design method?
- ❑ 7. Have proven CASE tools been chosen to assist in the software design?
- ❑ 8. Does your development team have experience with the chosen CASE tools? If not, has the schedule been adjusted to allow for learning to use the CASE tools?
- ❑ 9. Has an appropriate programming language been chosen and do you know the reasons it was chosen?

- ❑ 10. Does your development team have experience with the chosen programming language? If not, has the schedule been adjusted to allow for learning the chosen programming language?
- ❑ 11. Is the development team sufficiently skilled and experienced in programming to properly and efficiently design, code, and test the software?

10.3.2 During Project Execution

- ❑ 12. Are your requirements complete, unambiguous, and agreed to by both developers and stakeholders?
- ❑ 13. Have you completed both system and functional specifications, and have they been reviewed and approved by stakeholders?
- ❑ 14. Is the development team familiar with or provided with the appropriate opportunity to become familiar with the operating system and system hardware?
- ❑ 15. Is a detailed software design being completed, reviewed, and approved before coding starts?
- ❑ 16. Is testing being properly implemented and satisfactorily completed at unit, integration, and system levels before acceptance testing?
- ❑ 17. Are human factors being considered sufficiently in the software design?

10.3.3 At Completion

- ❑ 18. Does the completed software correctly implement the design?
- ❑ 19. Does the software meet the requirements?
- ❑ 20. Does the software meet the users' needs?

10.4 References

[1] Michael Black, "Introduction to Software Engineering" Lectures: www.cs.brown.edu/courses/cs032/

[2] DoD 5000.2-R, Mandatory Procedures For Major Defense Acquisition Programs (MDAPS) And Major Automated Information System (MAIS) Acquisition Programs, April 5, 2002. Section 4.3.5: http://sw-eng.falls-church.va.us/dod5000-2.html

10.5 Resources

ASD(C3I) Memorandum, "Use of the Ada Programming Language," April 29, 1997. Factors to consider when choosing selecting a programming language: http://sw-eng.falls-church.va.us/oasd497.html

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "Getting Software Engineering into Our Guts": www.stsc.hill.af.mil/crosstalk/2001/jul/bernstein.asp
- "The Software Engineer: Skills for Change": www.stsc.hill.af.mil/crosstalk/2001/jun/cross.asp
- "The V Model: www.stsc.hill.af.mil/CrossTalk/2000/jun/hirschberg.asp

DeGrace, Peter and Stahl, Leslie, *Wicked Problems, Righteous Solutions: A Catalogue of Modern Software engineering Paradigms*, Yourdon Press.

Department of Energy (DOE) *Software Engineering Methodology*: http://cio.doe.gov/sqse/sem_toc.htm

Guide to Software Engineering Body of Knowledge: www.swebok.org

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, Chapter 11, OO-ALC/TISE, May 2000. Available for download at: www.stsc.hill.af.mil/gsam/guid.asp

Program Manager's Guide for Managing Software, 0.6, 29 June 2001: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Software Engineering Body of Knowledge:
www.sei.cmu.edu/publications/documents/99.reports/99tr004/99tr004abstract.html

Software Engineering Education websites: http://faculty.db.erau.edu/hilburn/se-educ/

Software Engineering Institute: www.sei.cmu.edu

Software Engineering Process Group, Tutorials: http://prg.cpe.ku.ac.th/developer/tutorial.html

Software Reality. Stories of software engineering mistakes: www.softwarereality.com

University of Michigan, Introduction of Software Engineering:
www.engin.umd.umich.edu/CIS/course.des/cis375.html

University of Toronto, Software engineering notes: www.cs.toronto.edu/~sme/CSC444F/

Verification, Validation and Evaluation of Expert Systems: www.tfhrc.gov/advanc/vve/cover.htm

This page intentionally left blank.

Chapter 11

Assessing Project Health

CONTENTS

This page intentionally left blank.

Chapter 11

Assessing Project Health

"I see dead people… they're everywhere. They walk around like everyone else. They don't even know that they're dead." – from the movie "Sixth Sense"

11.1 Introduction

Projects can be viewed as living entities, requiring inputs to keep alive and having a purpose of producing something. In most organizations the project must have a charter or some other document to validate its existence and give it legal standing within the organization. If something can be considered a living entity, it can die. It also has health, or a status of its existence, indicating how well it is fulfilling its purpose or how close it is to death.

If a project is moving steadily towards its goals, according to the project plan, it can be considered healthy. The converse is also true. This, of course, assumes that the plan is realistic and based on historical data. Good health leads to life. Likewise, a project that is over budget, behind schedule, producing a poor quality product, or not meeting requirements can be diagnosed as having poor health. Poor health, if not checked and reversed, leads to early death. Some projects are dead long before anyone knows it.

Knowing whether your project is in good or poor health is absolutely essential to managing it. Hence, once a project has been given life, the first order of business for the Project Manager (PM) is to ascertain and track the project's health. This chapter discusses indicators and processes of assessing project health.

11.2 Process Description

Project health, like a human's health, can only be determined by observing and measuring specific indicators. There are four primary activities which gather information about project health. These are shown in Figure 11-1.

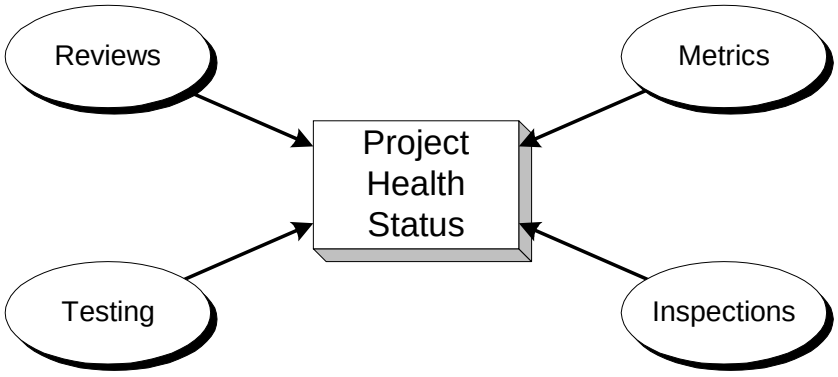

Figure 11-1 Activities for Assessing Project Health Status

These activities vary widely in their approach and are not all useful at the same time in the project life cycle. Reviews are snapshots of project status at specific times. While they can present trends by incorporating information from one or more of the other activities, by themselves, they provide a static look at the project. Inspections can be performed throughout the project whenever there is something, documents or otherwise, to inspect. Metrics can also be gathered throughout the project, depending on what is being measured. Data from both metrics and inspections can be used to spot trends and point to probable outcomes before they actually happen. Testing occurs toward the end of a project or development cycle and indicates quality and error content.

Wise project management will incorporate all these methods in assessing health throughout the project, recognizing their strengths and keeping in mind their weaknesses.

11.2.1 Indicators of Project Health

Just as a doctor will check pulse, blood pressure, temperature, and many other vital signs to determine a person's state of health, the activities introduced above examine or evaluate the life signs of a project. While there are many things that can be evaluated or monitored, the following list contains some of the more important project vital signs.

- Cost & Budget
- Schedule
- Critical Path
- Risks
- Funding
- Planning
- Requirements
- Quality
- Scope Stability
- Project Communications
- Human Resources
- Technical Progress

A proper state for each of these is vital to success and has an impact on the project as a whole. Any program of project health assessment should include these as a minimum.

11.2.2 Metrics

Metrics are measures of performance, quality, progress, or deviation from the plan. They can be used to evaluate health in any area. Metrics should be selected to indicate when and how progress is be made, and also illuminate areas where there are problems or deficiencies. A good starting point would be to implement a metrics program that monitors the indicators in the previous section. Chapter 8 of this book provides an overview to establishing a metrics program.

11.2.3 Reviews

Reviews are usually structured, scheduled evaluations of a project's progress and vital signs. Project reviews are often associated with project milestones and may be decision points, where the stakeholders or management determine whether to move to the next phase of the project. Other reviews may consider only a portion of the project, such as the software, technical progress, or test readiness. Reviews have many purposes. Some of the more common are listed here.

1. Help reduce cost, shorten schedules, and reduce defects. [1]
2. Provide training and experience by introducing team members to others' work and work styles. [1]
3. Discover and bring management attention to problems and issues.
4. Verify that the deliverables are meeting standards. [1]
5. Ensure everything is ready for the next project phase.
6. Demonstrate progress to stake holders and verify that development is on track.
7. Help identify possible malicious code and prevent its incorporation into the system. [1]

While reviews can evaluate almost anything, and be performed by any particular group, most reviews will be one of four types: [1]

1. Milestone Decision Review
2. Technical Review
3. Independent Expert Program Review
4. Walkthrough

The software development plan should also be reviewed at critical points to confirm, or revise if necessary, project costs, schedule, staffing, scope, etc.

11.2.3.1 Milestone Decision Review (MDR)

Milestone Decision Reviews are reviews of technology projects or acquisition programs, performed by the Milestone Decision Authority (MDA) at each milestone and other points in the project desired by the MDA. It usually consists of the presentation of program documentation and progress reports in an MDR briefing, but may also include demonstrations and inspections. When the milestone is achieved and the decision is made to proceed, an Acquisition Decision Memorandum (ADM) or equivalent document is signed. The MDR process is shown in Figure 11-2.

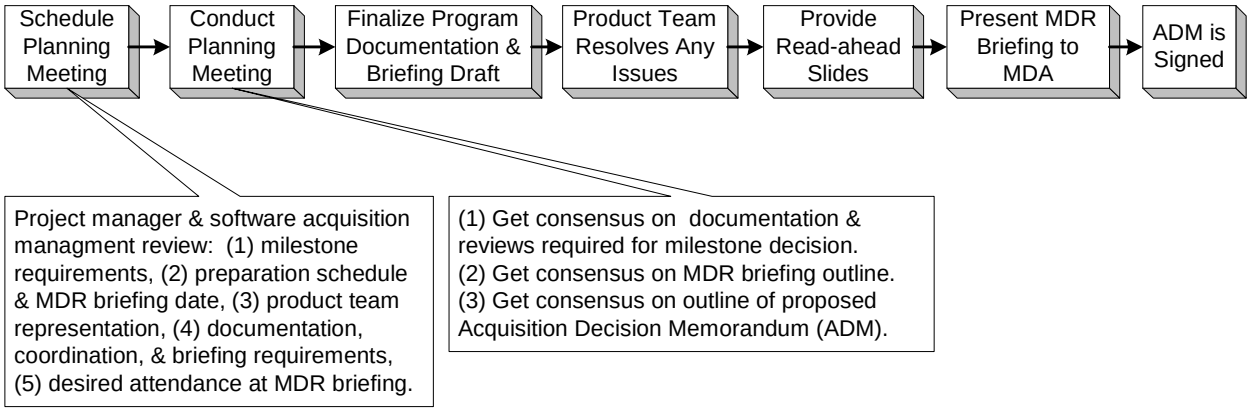

Figure 11-2 Milestone Decision Review Process

11.2.3.2 Technical Review

Technical Reviews, also called In-Process-Reviews (IPRs), are used to evaluate potential software or other products to determine if they meet the following criteria:

- Are suitable for their intended use.
- Meet project requirements.
- Function efficiently and effectively for the users and the mission.

Figure 11-3 depicts the technical review process.

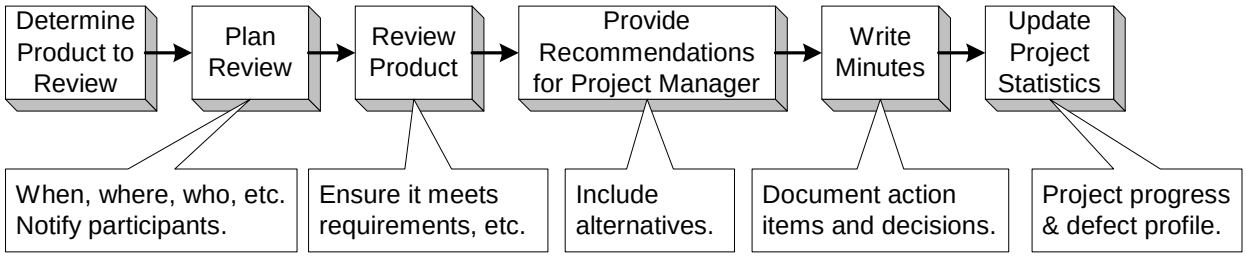

Figure 11-3 Technical Review Process

11.2.3.3 Independent Expert Program Review

The Independent Expert Program Review is considered a best practice by both defense and commercial industries. In these reviews Subject Matter Experts (SMEs) are brought in from outside the project and stakeholder community to review the project for adequate progress and adherence to standards and best practices. The process is shown in Figure 11-4.

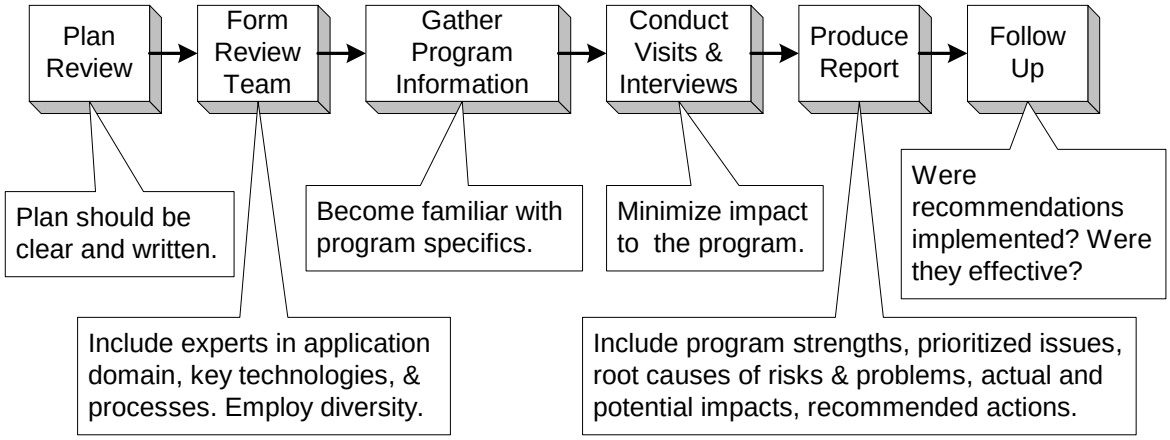

Figure 11-4 Independent Expert Program Review Process

11.2.3.4 Walkthroughs

Walkthroughs are used extensively in software development but are not limited to that discipline. As the name implies, a well-prepared individual "walks through" the product, explaining the general concepts to the audience. All parts of the subject software's requirements, design, and code are subject to walkthroughs. To quote Freedman and Weinberg, "Walking through the product, a lot of detail can be skipped – which is good if you're just trying to verify an overall approach or bad if your object is to find errors of detail." [3] The walkthrough process is shown in Figure 11-5.

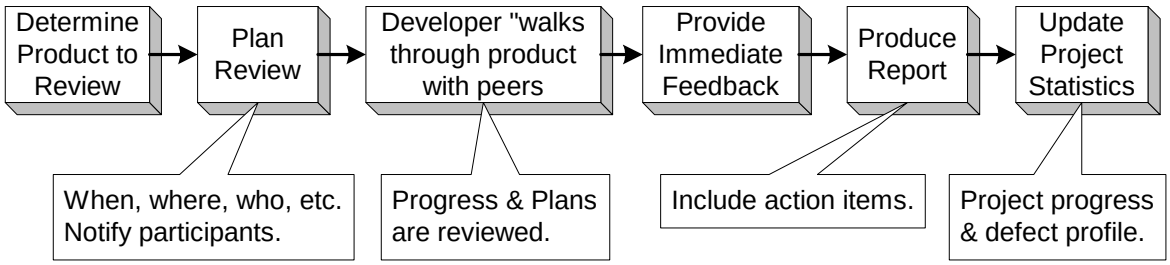

Figure 11-5 Walkthrough Process

11.2.4 Inspections

Another method of assessing project health, as well as improving product quality, costs, and schedule is the use of inspections. Inspection is the process of having individuals visually examine project products and documents, looking for errors and problems. Inspection can begin as soon as there is documentation to look at and can continue until the final product is delivered. Inspections, often implemented through the use of Independent Verification and Validation (IV&V) teams or via peer reviews, have proven their usefulness by discovering defects or design problems before they are implemented, shortening delivery time by reducing the time spent in integration and test phases. [2]

A special benefit of inspections is the fact that they begin to uncover errors long before testing can show the problem. A large German company found that, in comparison with defects found by formal inspection, defects discovered in testing cost 14.5 times as much to rectify, and those discovered by the customer cost 68 times as much. An IBM report placed the cost of fixing an error after release to be 45 times the cost to correct it during design. [2]

A final advantage gained by inspections is the data that can be collected from inspections and used to identify the more common problem areas, determine their causes, and change the development process to reduce or prevent those errors.

What should be inspected? While software source code is usually on the list, inspections should cover far more, beginning with requirements. Some of the better candidates for inspection materials are listed below.

- Requirements Specifications
- System documentation
- Design Documents
- Models
- Test Plans and Procedures
- Software Documentation
- Source Code

These are only basics. Essentially, any human-readable product involved with the development effort should be considered for inspection. [2] Those items produced earlier in the project, such as requirements, have the potential for greater payoff because they uncover defects earlier.

Unfortunately, good things do not come without cost. Inspections can cost from 5% to 15% of the project budget. You will need to determine what type and level of inspections are cost effective. This is done by analyzing the errors and defects found through inspection and realistically estimating what the cost to the project in time and money would have been had they been discovered later through other methods. When the probable cost of the errors has been found, this is compared with the cost of incorporating inspections. Money may not be the top consideration here. It may be that the savings in time far outweighs the cost of inspections.

The inspection process is simple and shown in Figure 11-6. After the initial inspection plan identifying who, what, when, and where, the cycle of inspect - report - statistics continues as each new artifact to be inspected is produced, until the end of the project.

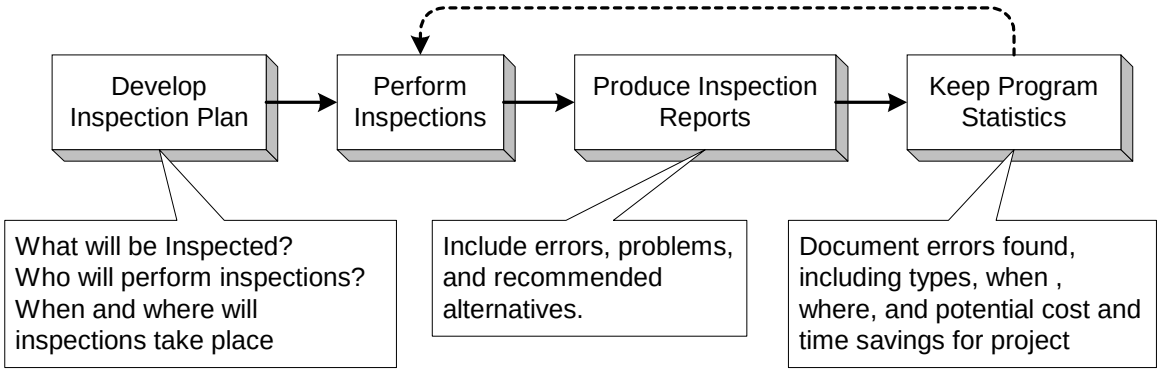

Figure 11-6 Inspection Process

11.2.5 Testing

Testing has two major purposes: to make a judgment about quality or acceptability, and discover problems or errors. The primary role of testing in assessing project health is to evaluate the quality of the products, their adherence to standards, functionality, and their compliance with requirements. As such it requires some product to test. This usually places testing in the latter part of the project. Because it works with the actual product, testing is an excellent indicator of health with reference to the product late in the project. On the down side, it is not useful for indicating health in the earlier phases of the project. Hence the importance of thorough test planning as early in each stage of the project as possible. By the time testing uncovers problems, significant rework will be required to correct defects, as shown in Figure 11-7.

"Inspections find the defect, while testing – which usually occurs one or more development phases after the opportunity to inspect has passed – finds only the symptom." – Priscilla J. Fowler

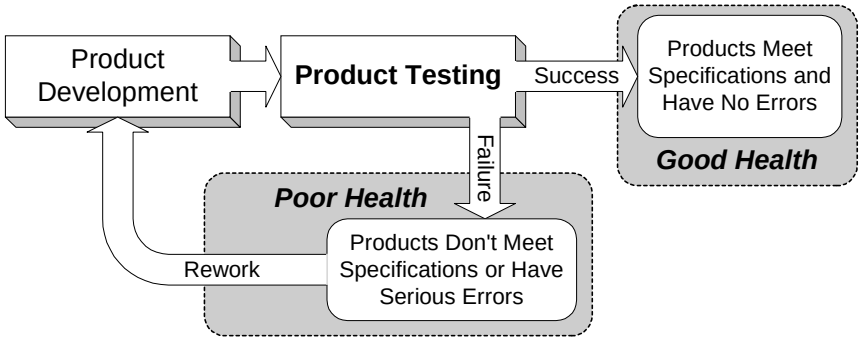

Figure 11-7 Role of Testing in Assessing Project Health

It can be seen that while testing is important, it comes too late to rely on as an early warning indicator. An overview of testing can be found in Chapter 12 of this handbook.

11.2.6 Summary

By using a combination of reviews, metrics, inspection, and testing, a project's progress or deviation from the planned baseline can be accurately followed and even predicted. This will allow you to proactively manage the project, avoiding or dealing with problems before they slow the project's progress or become showstoppers.

Carefully plan when, where, and how you will use the assessment methods. In addition to constantly monitoring your project's health, regularly evaluate your health assessment program for adequateness and efficiency. Don't let the assessment program adversely affect the project it is monitoring. Make changes as necessary and keep a record of what works well and what doesn't for future reference. Get help from experienced PMs in implementing your program. Use the resources at the end of this chapter to get more information of implementing software-specific metrics and assessments.

11.3 Project Health Assessment Checklist

This checklist is provided to assist you in monitoring the health of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

11.3.1 Before Starting

- ❑ 1. Have you prepared an overall plan for assessing project health?
- ❑ 2. Does your plan include a documented assessment process answering who, what, when, where, and how?
- ❑ 3. Does your assessment plan incorporate reviews, metrics, inspections and testing?
- ❑ 4. Do you have planned baseline budgets, schedules, etc. to compare actual project status to?
- ❑ 5. Have you scheduled regular reviews of your assessment process?
- ❑ 6. Do you have a database prepared to record project status for historical purposes?
- ❑ 7. Have you minimized the interference of your assessment activities with the project as much as possible?

11.3.2 Metrics

- ❑ 8. Have you developed a metrics plan as outlined in Chapter 8?
- ❑ 9. Have you selected appropriate measures and metrics for the different areas of your project (e.g. Have you implemented software development metrics applicable to your particular software efforts?)
- ❑ 10. Have you used historical data from other projects in establishing your metrics program?

11.3.3 Reviews [1]

- ❑ 11. Are reviews included in the project plan and schedule?
- ❑ 12. Is there a written plan for each review?
- ❑ 13. Does the plan define the scope of the review?
- ❑ 14. Does the plan specify how the review results will be documented and reported?
- ❑ 15. Does the plan identify the data to be collected?
- ❑ 16. Has the review been planned to minimize project impact?
- ❑ 17. Is a review follow up scheduled?
- ❑ 18. Does the report include recommended actions?
- ❑ 19. Are defects documented and tracked to closure?
- ❑ 20. Are all milestone stakeholders represented?
- ❑ 21. Are the Integrated Product Teams (IPTs) appropriately represented?
- ❑ 22. Does at least part of the technical review focus on software development and management?
- ❑ 23. Is the software product developed by the project suitable for its intended use?
- ❑ 24. Has risk management been addressed?
- ❑ 25. Have security issues been addressed?
- ❑ 26. Does the software comply with required standards and regulations?

11.3.4 Inspections

- ❑ 27. Do you have a plan for what is to be inspected, when, and by whom?
- ❑ 28. Have you used historical data to determine what is cost-effective to inspect and what is not?
- ❑ 29. Do your inspections minimize interference to the development work?
- ❑ 30. Have you emphasized inspections in the earlier stages of the project where testing cannot be performed?
- ❑ 31. Do you use the data collected from inspections to correct errors to keep them from propagating into later development stages?
- ❑ 32. Do you keep a history or database of all inspection activity, findings, and effectiveness?

11.3.5 Testing

- ❑ 33. Do you have a comprehensive test plan for your project, designating who, what, when, where, and how?
- ❑ 34. Have you made testing considerations a major input in the early stages of development?
- ❑ 35. Do you understand the limits of testing?
- ❑ 36. Have you implemented a testing program as outlined in Chapter 12?
- ❑ 37. Are you keeping a history of testing plans, results, and effectiveness?

11.4 References

[1] *Program Manager's Guide for Managing Software*, 0.6, 29 June 2001, Chapter 9: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

[2] Wiegers, Karl E., "Improving Quality through Software Inspections,": www.processimpact.com/pubs.shtml

[3] Friedman, Daniel P. and Gerald M. Weinberg, *Walkthroughs, Inspections, and Technical Reviews,* Dorset House Publishing, New York NY, 1990

11.5 Resources

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "The Determining Factor": www.stsc.hill.af.mil/crosstalk/2000/may/dynes.asp
- "Help Identify and Manage Software and Program Risk": www.stsc.hill.af.mil/crosstalk/2000/nov/baldwin.asp
- "Software Mini-Assessments: Process and Practice": www.stsc.hill.af.mil/crosstalk/1999/oct/natwick.asp
- "How Much Code Inspection is Enough": www.stsc.hill.af.mil/crosstalk/2001/07/index.html
- "Using Inspection Data to Forecast Test Defects": www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1998/05/inspection.asp
- "Document Diseases and Software Malpractice": www.stsc.hill.af.mil/crosstalk/2002/nov/daich.asp

Department of Energy (DOE) *Software Engineering Methodology*: http://cio.doe.gov/sqse/sem_toc.htm

Ebenau, Robert G. and Susan H. Strauss, *Software Inspection Process*, McGraw-Hill, Inc., New York NY, 1994

Friedman, Daniel P. and Gerald M. Weinberg, *Walkthroughs, Inspections, and Technical Reviews*, Dorset House Publishing, New York NY, 1990

Gilb, Tom and Dorothy Graham, *Software Inspection*, Addison-Wesley, New York NY, 1993

Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM), Version 3.0, Chapter 13 and Appendix N, OO-ALC/TISE, May 2000. Available for download at: www.stsc.hill.af.mil/gsam/guid.asp

Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapter 9: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Radice, Ronald A., *High Quality Low Cost Software Inspections*, Paradoxicon Publishing, Andover MA, 2002

U.S. Treasury Dept., *Systems Development Life Cycle Handbook*, Chapters 3 and 4: www.customs.ustreas.gov/contract/modern/sdlcpdfs/

Wiegers, Karl E.,

- *Peer Reviews in Software: A Practical Guide*, Addison-Wesley, 2002
- "Seven Truths About Peer Reviews": www.processimpact.com/pubs.shtml
- "Do your Inspections Work?": www.stickyminds.com/sitewide.asp?Function=WEEKLYCOLUMN&ObjectId=3526&ObjectType=ARTCOL
- "A Software Metrics Primer": www.processimpact.com/pubs.shtml
- "Metrics: Ten Traps to Avoid": www.processimpact.com/pubs.shtml#metrics
- "Lessons from Software Work Effort Metrics": www.processimpact.com/pubs.shtml#metrics
- "Improving Quality through Software Inspections,": www.processimpact.com/pubs.shtml
- "Seven Deadly Sins of Software Reviews,": www.processimpact.com/pubs.shtml
- "When Two Eyes Aren't Enough,": www.processimpact.com/pubs.shtml

Chapter 12

Testing

CONTENTS

This page intentionally left blank.

Chapter 12

Testing

"We test because we know that we are fallible …" – Paul C. Jorgensen

12.1 Introduction

Testing is not new to any of us. It comes almost daily and can be formal or informal. Our steps through school and training are marked by tests. For some, each day brings a test of survival. Progress is never realized without some form of testing. This maxim also holds true for software development projects.

While failure to properly implement a functional test program for software used by a commercial enterprise can lead to loss of revenue and the demise of the company, failure in a military environment can lead to mission failure, precipitate injury and death, and ultimately jeopardize the survival of the nation. This chapter discusses testing principles and processes, particularly as they relate to software testing. However, the concepts can also be readily applied to other types of systems. Because testing is part of a comprehensive assessment and evaluation program, you are encouraged to also read Chapter 11, "Assessing Project Health," if you have not already done so.

A good testing program is essential for reliable operational performance, and will significantly reduce support and maintenance costs. Properly implemented testing significantly improves the probability of project success, thereby enhancing the likelihood of mission success. Test planning and preparation impact the project early by helping define requirements that are testable. As products are being developed, testing allows the developers to discover and fix problems before they can become showstoppers. While a good testing program costs money and can't necessarily put a troubled project on the road to recovery by itself, it does provide extra insurance that most project managers need by helping to signal and avoid trouble. [1]

Testing has two fundamental purposes: [2]

1. To evaluate quality or acceptability of that which is being tested.
2. To discover problems or errors.

These primary purposes can be further expanded into three groups of objectives, shown in Table 12-1.

Table 12-1 Objectives of Testing [1]

Demonstration	Detection	Prevention
• Show that the system can be used with acceptable risk. • Demonstrate functions under special conditions. • Show that products are ready for integration or use.	• Discover defects, errors, and deficiencies. • Determine system capabilities and limitations. • Determine quality of components, work products, and the system.	• Provide information to prevent or reduce the number of errors. • Reduce the number of early errors propagated through to later phases. • Clarify system specifications and performance. • Identify ways to avoid risks and problems in the future.

12.2 Process Description

Testing is one of the major inputs to project health, serving not only as an indicator of, but also as a primary contributor to, the health of the project by improving quality and discovering errors and risks before they become critical problems. Because a testing program is a major part of the project it should be well defined and documented. It should be considered in and influence every major phase of the project, as shown in Figure 12-1.

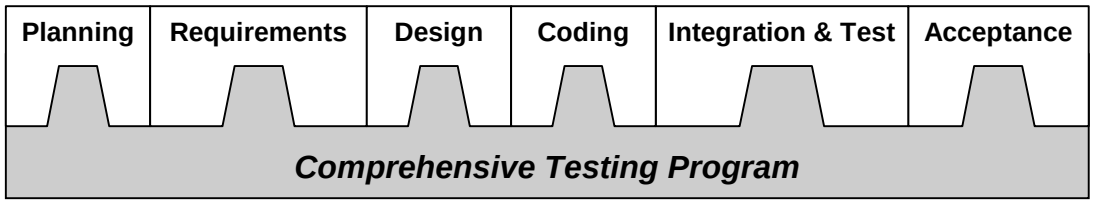

Figure 12-1 An Effective Testing Program is Part of all Project Phases

Testing is accomplished by implementing a testing program. A good program includes the right people, methods, tools, and equipment needed to perform the testing, along with a testing process to integrate all these into an efficient, effective program, as shown in Figure 12-2. [3] Testers should be trained, skilled, and motivated people. Employing people who lack these attributes will increase the amount of time and resources needed for testing and increase the risk of overlooking important testing considerations.

Because testing is an integral part of the development process, adequate time and resources are reserved for testing. Tests also serve as milestone and progress indicators. Proven methods should be used to facilitate efficient, effective testing. It must be remembered that everything cannot be tested. Even if it were possible to think of all possible test cases, the effort would consume over 90% of a project's resources and take an unacceptable length of time. Because of this, testing must be risk-based, where those areas with the greatest risk of failure are given the most attention. Effective testing is creative and uses careful analysis, planning, and design to select test strategies, methods, and tests to get the greatest amount of evaluation for the testing resources available. [1]

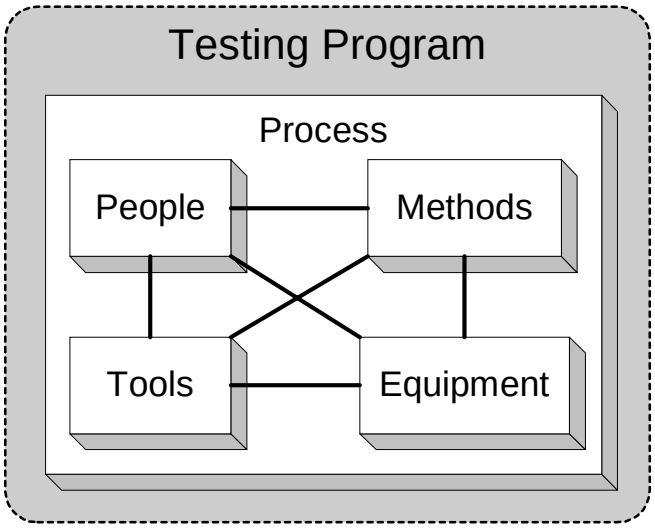

Figure 12-2 Elements of an Effective Testing Program

Appropriate equipment should be available and properly used to allow thorough testing. Without the right equipment, testers may have to use slower, roundabout testing methods, taking longer to test, perhaps missing important test results, or even bypassing important tests.

Test tools cover a broad spectrum of testing assistants. They range from simple databases to track tests, results, and requirements met, to full automatic testing systems that perform comprehensive testing, including varying the logical flow of testing, depending on test results. Tools may be hardware based, software based, or both. The choice of tools requires consideration be given to the type of testing, the amount of testing, testing expertise of the organization, costs, schedule, possibility for reuse, and requirements. A commonly used tool is a tracking system for recording what has been tested and what has not. Additionally, a database of test results is maintained for current and future reference. Test results are used as a feedback mechanism not only to determine which components need to be reworked, but for improving the development process and methods in the current and future projects.

Too often, testing is viewed as a hurdle to get over, rather than a tool for improving both product quality and the development process. When testing is considered and planned early in the project, along with establishing require-

ments that are testable, it can provide guidance throughout the project and reduce the time required for rework. Early life cycle testing helps prevent propagation of defects to later stages of development.

12.2.1 Establishing a Test Process

The testing process is the keystone to effective testing. It defines how test components are used with each other, and what is to be done at what times. The test process should be active throughout the entire development life cycle as indicated by Figure 12-1. It begins with planning and continues on through the acceptance of the developed product. Table 12-2 lists the major testing activities performed during each major phase of the project.

Table 12-2 Testing Activities of Project Phases

Phase	Testing Process Activity
Planning	Testing activities are scheduled throughout the project. Testing to meet requirements is established as a project priority.
Requirements	Requirements are written so that they are testable.
Design	System and components are designed to be testable and to meet requirements by passing tests.
Coding	Testing is performed in the code-test cycle (see Figure 12-6) and on complete software units. Debugging and regression testing is also performed.
Integration & Test	Testing is performed on integrated subsystems and systems (integration and system testing.) Debugging and regression testing are also performed. (See Figures 12-3 and 12-6.)
Acceptance	User acceptance testing is performed.

12.2.1.1 Testing Levels

A testing process should also follow a hierarchy of testing (shown in Figure 12-3) where components are tested at the lowest level and then at successively higher levels of integration until the complete system is tested and debugged. Finally acceptance testing is performed.

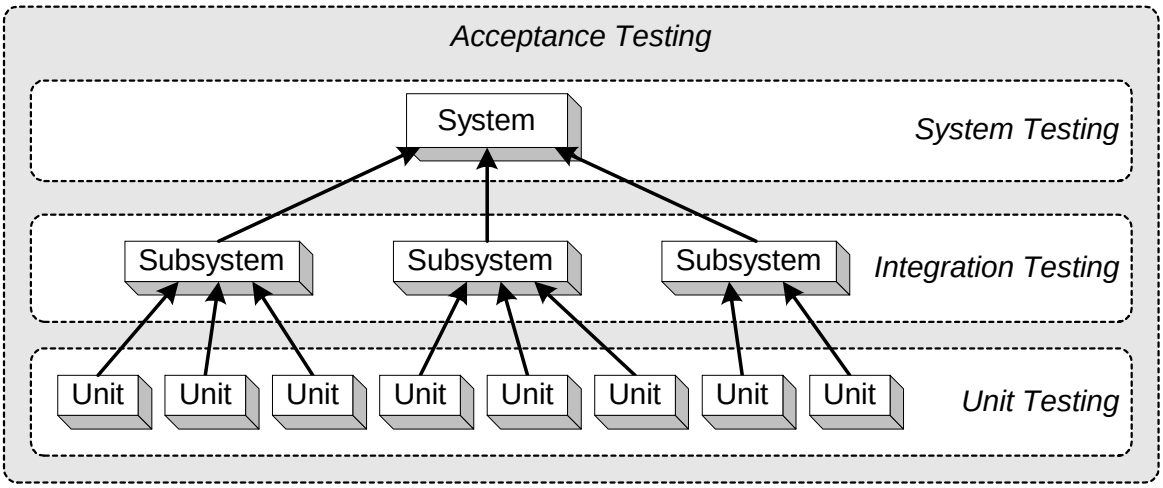

Figure 12-3 Testing Hierarchy

12.2.1.2 The Testing Process

The overall test process is shown in simplified form in Figure 12-4. Because it has already been discussed, the process leaves out the consideration of testing in establishing requirements and in design. After completion of the first (planning) activity, all other activities are repeated for unit, integration, and system testing. Because of the code-test-debug cycle (shown in Figure 12-6), there may be several iterations of coding and integration, along with their accompanying testing.

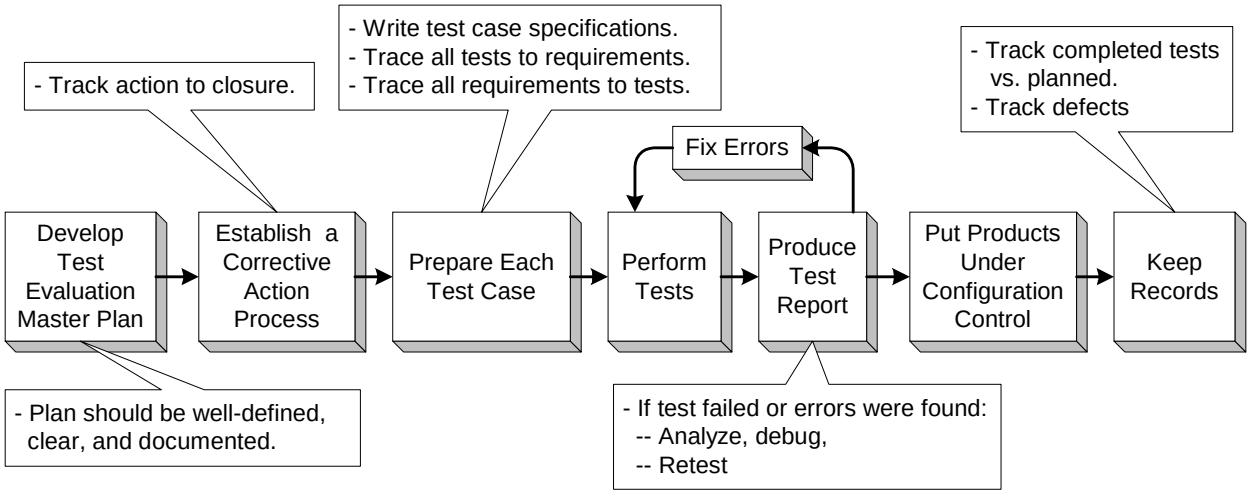

Figure 12-4 Simplified Testing Process

12.2.1.3 Test Planning

When planning the testing program, a hierarchy of plans is involved in the process, each providing test direction at different levels of the system and detail. Figure 12-5 depicts the Test Planning Tree as defined by the Software Program Managers Network (SPMN). The contents of these planning documents which relate to testing are summarized in Table 12-3. [4] More can be found on this and other testing principles in the SPMN's highly recommended *Little Book of Testing*, Volumes I and II. [1]

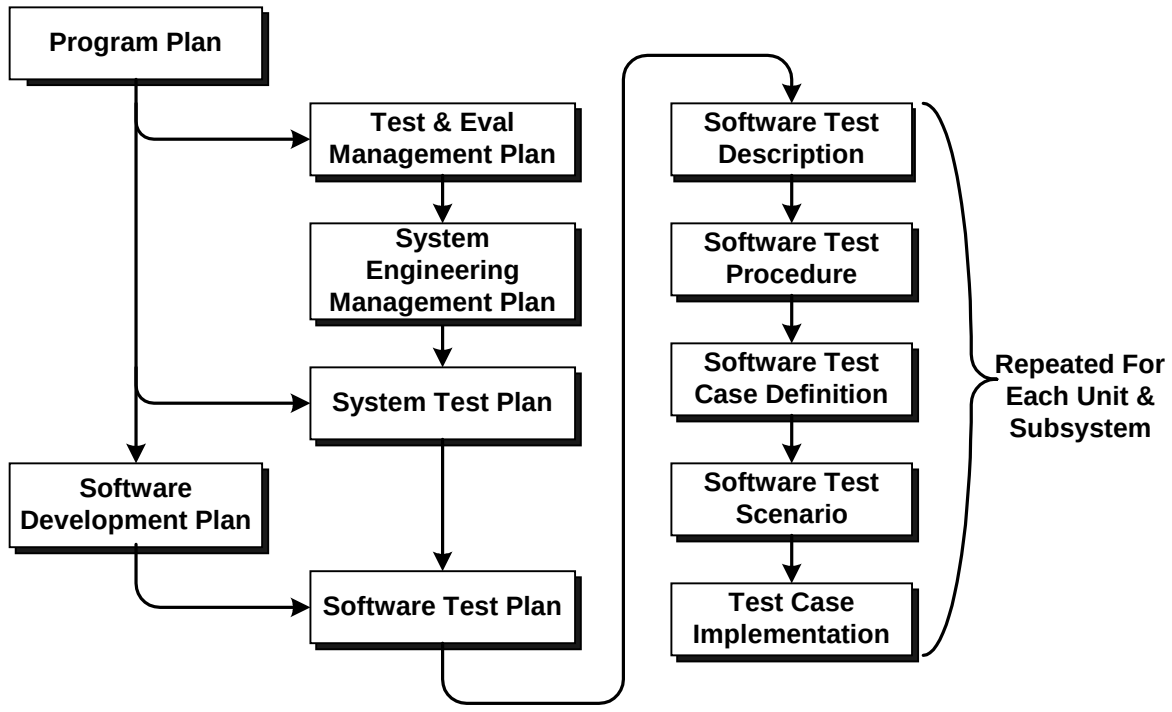

Figure 12-5 Test Planning Tree [4]

Several of these documents may be combined where appropriate. Often, the test procedure will contain the information for the case definition, test scenario, and case implementation.

Table 12-3 Testing Related Content of Project Planning Documents [4]

Planning Document	Testing Related Content
Program Plan	Defines program requirements, strategies, etc. Identifies test requirements.
Test & Eval Management Plan	Establishes entire testing strategy and practices.
System Engineering Management Plan	Identifies plans, methods, standards, processes, and systems for integration and test requirements.
System Test Plan	Defines system testing strategies, controls, and processes. Test case requirements are discussed and success criteria are identified.
Software Development Plan	Describes the criteria for overall integration, test, and completion.
Software Test Plan	Defines software and integration test plans, strategies, controls, and processes. Also defines test case requirements and completion criteria.
Software Test Description	Describes the requirements for individual test cases.
Software Test Procedure	Provides a step-by-step procedure for executing the test. It traces to a requirement specification.
Software Test Case Definition	Defines the test case and specifies success criteria for a test step.
Software Test Scenario	Identifies specific data, sources, interfaces, timing, expected response, and the next step to take in response to a test condition.
Test Case Implementation	Executable test case.

12.2.2 Types of Testing

Many different types of testing are used to fulfill the many different testing needs. Several of the major types are summarized here. Figure 12-6 indicates the phase of the project where some of these are generally found. Note that test types may be found in more than one phase. Remember that maintenance projects are usually considered as development projects also, ending with special maintenance and regression testing.

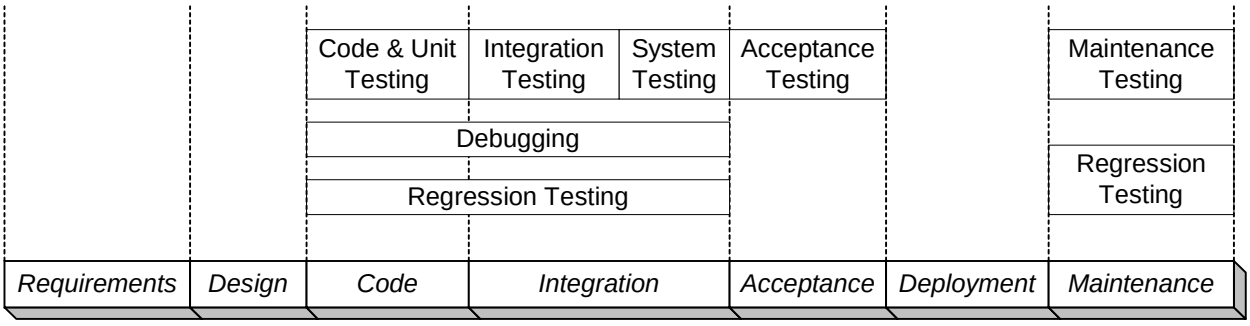

Figure 12-6 Types of Testing Performed During Different Project Phases

12.2.2.1 Debugging

Debugging is testing used from the unit to the system level to determine what is causing errors. It consists of search methods used to isolate problems to a specific module or cause. When the problem is found it is fixed and retested. While a formal test failure is usually an indicator of errors, debugging often involves a great deal of "free-form" testing by software and systems engineers. While training can help, most expertise in debugging comes from experience.

12.2.2.2 Code and Unit Testing

As a software engineer codes a software module or unit, there will usually be testing of different pieces along the way to make sure they are accomplishing the purpose of the unit. This is code testing and will consist mostly of informal, free-form testing. Unit testing is performed when the unit is believed to be complete. It is tested to make sure all inputs are handled correctly, producing the correct outputs with the proper timing, etc. Unit testing is usually more formal because the unit will probably need to handle specific inputs and produce specific outputs or actions. When units are complete, they are ready for the integration process.

It should be noted that unit testing is not always as straightforward as the name implies. Each unit should be well defined before the testing is defined. Not every module or source file is necessarily a unit, especially when considering code maintenance or enhancement. A unit may consist of a single module, part of one module, or may be comprised of several modules and their associated files. A unit may overlap several requirements, or a single requirement may involve several units. A system architect may be necessary to assist the test definition process.

12.2.2.3 Integration Testing

While software modules may function well by themselves when they are developed, getting them to work together efficiently and correctly is another matter. After they have been coded and tested individually, individual software components are combined to form a final software product. During this integration effort, tests are performed on various groupings of components to determine how well they work together. Incompatibilities, errors, and inadequacies are discovered and fixed. Eventually, all software modules are integrated and debugged so they function correctly as a whole.

12.2.2.4 System Testing

When the software, hardware, and other subsystems are complete, they in turn are integrated and tested as a system. This is the final development testing. Any problems or errors discovered during systems testing are analyzed to determine which subsystems are at fault, then those subsystems are sent back for debugging, with its attendant code, unit, and integration testing. Figure 12-7 shows the various levels of testing associated with development and how problems and errors feed back to earlier developmental stages. System testing evaluates the functionality of the system, including capabilities, compatibility, stability, performance, security, and reliability. [5]

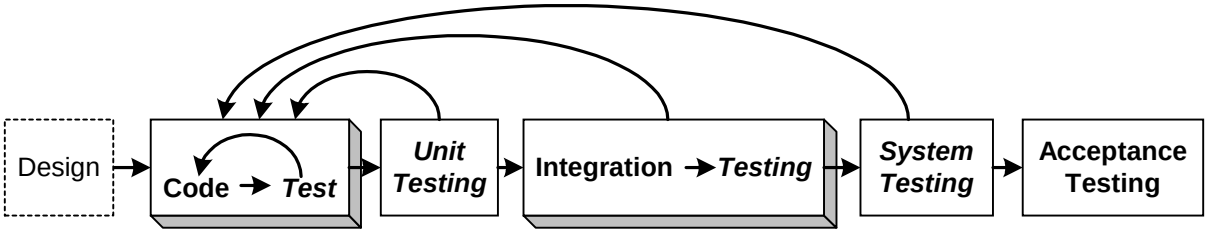

Figure 12-7 Development, Test, and Debug Cycle

12.2.2.5 Acceptance Testing

Acceptance testing is that formal demonstration that the system performs according to requirements. It should be rehearsed during the system testing phase so that there are no failures or problems to be discovered. Acceptance testing is usually witnessed by the customer and other stakeholders. A failure at this point is usually indicative of incomplete testing in the development phase. Because the format and procedures of the acceptance tests have been coordinated and agreed upon among developers and stakeholders, successful completion of the test should signal acceptance by the customer and clear the way for deployment.

12.2.2.6 Regression Testing

Regression testing is not associated with any particular stage of the project, but should be performed whenever there has been a change to a component or the system. It consists of testing components or a system after changes have been made to verify that the components or systems still comply with the requirements and that the modifications have not caused unintended effects. [5]

12.2.2.7 Maintenance Testing

Maintenance testing must be performed anytime there is a maintenance upgrade to the system. Its purpose is to ensure the new modifications are properly integrated and work with the rest of the system. It also verifies that the upgrade provides the additional functionality the maintenance upgrade is supposed to add. Maintenance testing should include regression testing to ensure the upgrade does not cause any undesired effects. [5]

12.2.2.8 Alpha Testing

Alpha testing is a preliminary field test by a select group of users with the purpose of finding bugs not discovered through previous testing and to refine the user interface. It is an extension of system testing and may or may not be used, depending on the project and product. The product is complete by this time but not necessarily refined. The test group is usually made up of people within the developing organization, but not the developers themselves. [5]

12.2.2.9 Beta Testing

Beta testing is similar to and performed following alpha testing. Like alpha testing, it is optional, depending on the project. The key difference is that the testers consist of selected users outside the developer's organization. [5]

12.2.2.10 Black box testing

Black box testing is testing the function of a component or system from a user's point of view without regard to the internal structure or logic involved. [5] This should be done at various times throughout the development to maintain an understanding of the user's perspective and meet the user's needs as well as the requirements.

12.2.2.11 Stress Testing

Stress testing is a form of system testing. The systems or subsystem is tested under extreme or abnormal conditions outside the operational envelope with the purpose of finding the limits where the item being tested fails or breaks down. This enables the testers to determine how much margin there is between expected operating conditions and failure conditions. Stress testing may also be used to determine sensitivity, which types of conditions or combinations thereof affect the system most and least. [6]

Stress testing of hardware may include vibration, pressure, temperature, and other environmental conditions. In software systems conditions may include abnormal quantities, high frequency of interrupts, etc.

12.3 Testing Checklist

This checklist is provided to assist you in understanding the testing issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

12.3.1 Before Starting

- ❑ 1. Is testing planned for and considered throughout the entire development life cycle?
- ❑ 2. Is the overall testing strategy defined and documented, and is it an integral part of and consistent with the development program? [4]
- ❑ 3. Is the testing process well defined, documented, understood, and supported by the development team and management?
- ❑ 4. Are test requirements clearly defined? [4]
- ❑ 5. Are test methods, techniques, controls, and standards clearly defined and consistent with the testing strategy? [4]
- ❑ 6. Is each test activity traceable to specific requirements? [4]
- ❑ 7. Are configuration management and quality assurance in place and are they adequate to support the testing strategy? [4]
- ❑ 8. Are testers trained, skilled, and motivated people?
- ❑ 9. Have adequate time and resources been reserved for testing?

- ❑ 10. Are time and resources allocated for test preparation early in the project life cycle?

12.3.2 During Execution

- ❑ 13. Is testing used as a primary tool to ensure good project health?
- ❑ 14. Is testing implemented as a tool for improving product quality and the development process as a whole?
- ❑ 15. Is early life cycle testing used to prevent propagation of defects to later stages of development?
- ❑ 16. Is a tracking system being used to record what has been tested and what has not?
- ❑ 17. Is a database of test results being maintained for current and future reference?
- ❑ 18. Are tests used as milestone and progress indicators?
- ❑ 19. Is the right amount of testing being done to balance risk with available time and resources?
- ❑ 20. Are you using inspections and other evaluation methods (see Chapter 11) to reduce the errors found through testing?
- ❑ 21. Do you know when your testing is complete?

12.4 References

[1] Software Program Managers Network, *Little Book of Testing, Vol. 1*, 1998: www.spmn.com/products_guidebooks.html

[2] Jorgensen, Paul C., *Software Testing A Craftsman's Approach*, CRC Press, 1995, p.3.

[3] Kit, Ed, *Software Testing in the Real World*, Addison-Wesley, 1995, p.3.

[4] Software Program Managers Network, *Little Book of Testing, Vol. II*, 1998: www.spmn.com/products_guidebooks.html

[5] *Program Manager's Guide for Managing Software*, 0.6, 29 June 2001, Chapter 11: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

[6] University of South Australia, Software Testing notes: http://louisa.levels.unisa.edu.au/se1/testing-notes/testing.htm

12.5 Resources

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "The Problem with Testing": www.stsc.hill.af.mil/crosstalk/2001/07/index.html
- "Maintaining the Quality of Black-Box Testing": www.stsc.hill.af.mil/crosstalk/2001/05/korel.html
- "Proven Techniques for Efficiently Generating and Testing Software": www.stsc.hill.af.mil/crosstalk/2000/06/wegner.html
- " Model to Assess Testing Process Maturity": www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1998/11/burnstein.asp
- Planning Efficient Software Tests": www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1997/10/planning.asp
- "Using Statistical Process Control with Automatic Test Programs": www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1998/08/statistical.asp
- "Engineering Practices for Statistical Testing": www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1998/04/statistical.asp
- "Using Inspection Data to Forecast Test Defects": www.stsc.hill.af.mil/crosstalk/frames.asp?uri=1998/05/inspection.asp

Department of Energy (DOE) *Software Engineering Methodology*, Chapters 7-9: http://cio.doe.gov/sqse/sem_toc.htm

NASA, Recommended Approach to Software Development, Sections 7-9: http://sel.gsfc.nasa.gov/website/documents/online-doc.htm

Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapters 10-11: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Software Testing Stuff: www.testingstuff.com/

Software Testing Institute: www.softwaretestinginstitute.com/

SPMN Testing Guidebooks available for download at: www.spmn.com/products_guidebooks.html

SPMN 16 Critical Software Practices: www.spmn.com/16CSP.html

University of South Australia, Software Testing notes: http://louisa.levels.unisa.edu.au/se1/testing-notes/testing.htm

This page intentionally left blank.

Chapter 13

Systems Engineering

CONTENTS

This page intentionally left blank.

Chapter 13

Systems Engineering

"That central life is somewhat superior to creation, superior to knowledge and thought, and contains all its circles." – Ralph Waldo Emerson

13.1 Introduction

An arrow may seem like a simple object with a single, one-dimensional function – to kill or wound something. Nevertheless, if one were asked to make an arrow, many questions would suddenly become critical. How big should the feathers on the back end be? Are three feathers sufficient, or would four be better? Is there a better material than feathers? What should the length, diameter, strength, and weight of the shaft be? What kind of point should the arrow have? What is it to be used against? How should it interface with the target? (i.e. How far must it penetrate? How sharp should it be? Should it be bladed or round? Should it be made so that it cannot be withdrawn without tearing up the target?) What materials have worked best in the past? Now that the arrow is seen as a collection of components, how do the feathers and point connect to, or interface with, the shaft to form the arrow system? And finally, how does the arrow interface with the bow to form the larger system of "bow and arrow?"

Today's weapon systems are far more complex than the bow and arrow. But in like manner they are composed of systems of subsystems and components. In some cases, systems are even composed of other systems. Also in like manner, each subsystem must be analyzed to correctly specify its requirements, and then be designed so that it will meet the subsystem requirements, interface correctly with the other subsystems to form the system, and meet the system requirements. These are many aspects of systems engineering. But they are not all of systems engineering. Figure 13-1 shows a summary of the overall scope of this discipline that spans the product life cycle.

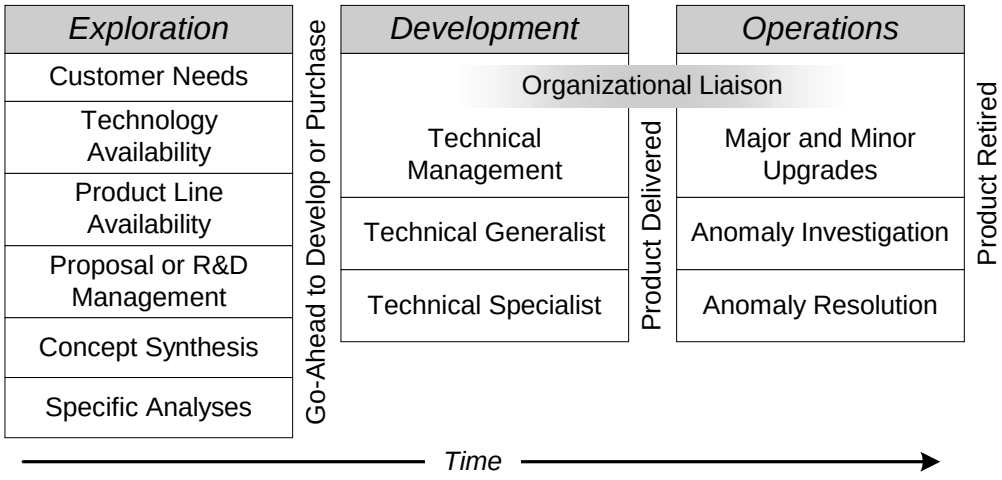

Figure 13-1 Scope of Systems Engineering [1]

Many of the other chapters in this book deal with project management principles and issues. By contrast, systems engineering consists more of technical principles and issues. While the overall project is directed by a Project Manager (PM), who directs the project toward project success, systems engineering is a technical discipline and should be led by a skilled and experienced Systems Engineer (SE), whose primary goal is product success. Systems engineering transforms an operational need into a set of system performance parameters and a preferred system configuration. [2]

When projects with technical problems are analyzed to determine the root of their trouble, the majority can trace the cause back to insufficient or improper systems engineering. The complexity of modern systems makes it an absolute necessity to divide the product into various segments and subsystems, employing many disciplines to design and

build these parts of the whole. However, the parts will not work together unless there is an effective system-level designer and builder ensuring all the parts interface, work together, and meet the requirements. Systems engineering defines the system requirements and specifications, designs the system, including allocating functionality to subsystems, and continues to monitor and assist product development until integration and testing are complete to make sure the system is built according to specifications.

13.2 Process Description

Systems engineering can be divided into two major sub-disciplines. The first is the actual technical knowledge domain, usually referred to *systems engineering*. The second is *systems engineering management.* Often, one or the other of these two is mistaken as the whole discipline. Systems engineering management consists of three primary activities, *development phasing*, *life cycle integration*, and the *systems engineering process*. These activities are shown, along with their products or sub-activities in Figure 13-2.

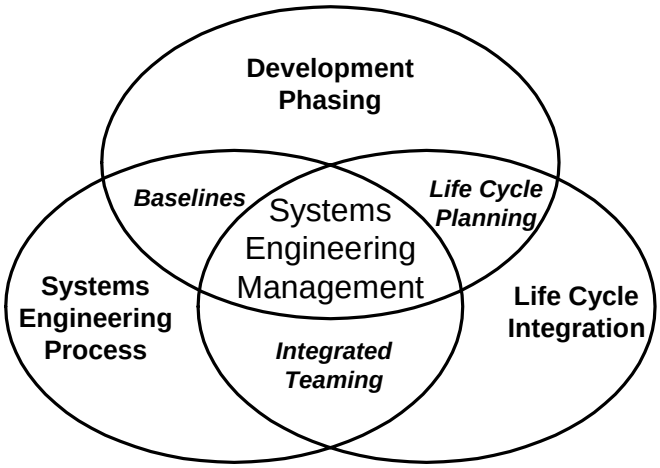

Figure 13-2 Systems Engineering Management Activities [2]

13.2.1 Development Phasing

The design process is a series of development stages where progressively more detailed systems descriptions or designs are produced. To begin, studies of what the system should do and how it would operate are performed and documented in the *system concept*. This is the first look at whether there is a feasible solution to the operational need. Assuming the conceptual system is technically, operationally, and economically feasible, development proceeds through system development to produce a *functional baseline*, where all the functions the system are identified and specified. The next step is the preliminary design, during which subsystems are defined and the functions are allocated to the various subsystems to create the *allocated baseline*. The final step in this process is the detailed design. During this phase actual products to perform the functions are identified for acquisition or development, producing the *product baseline*. This stepped process is shown in Figure 13-3. [2]

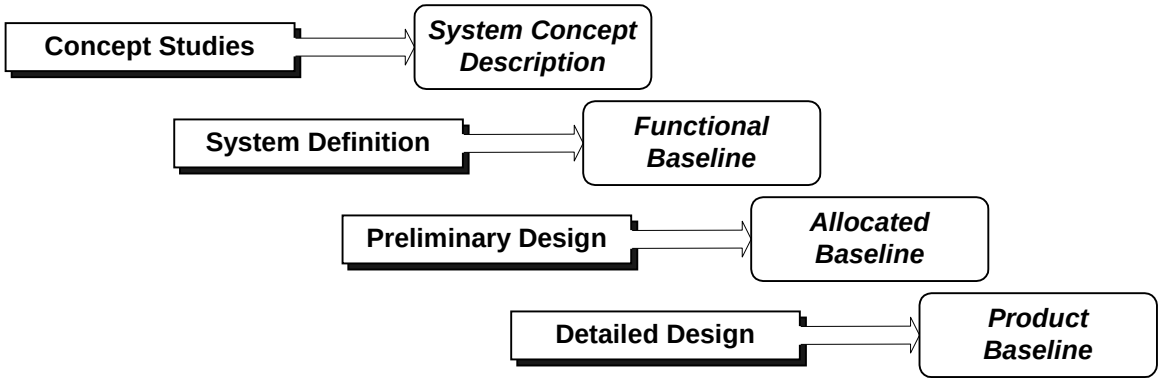

Figure 13-3 Steps and Products of Development Phasing

13.2.2 Systems Engineering Process

While development phasing determines what major design activities are performed, the actual process performed is known as the systems engineering process, shown in Figure 13-4.

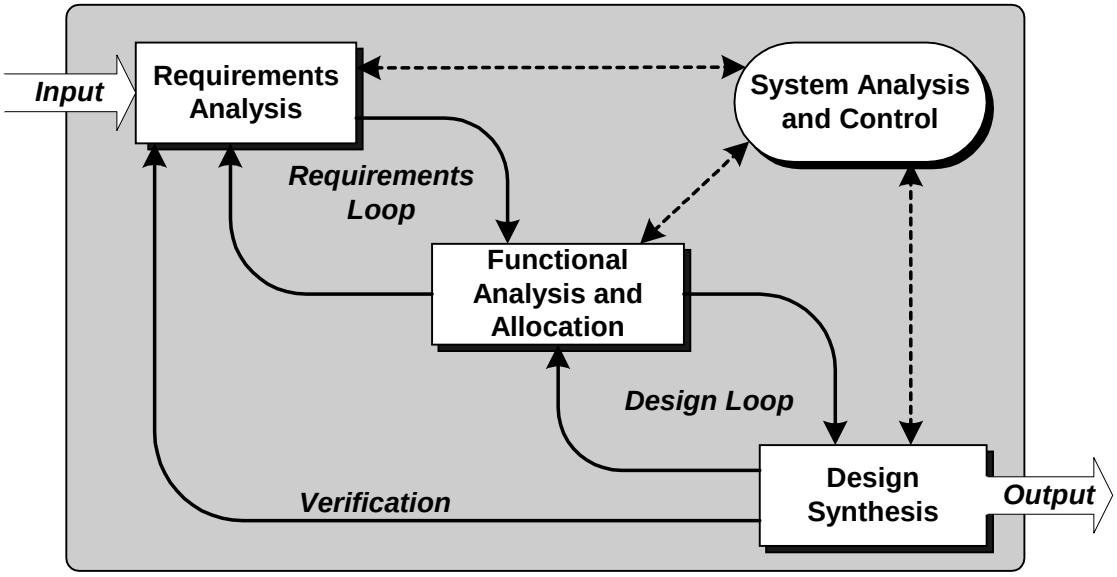

Figure 13-4 Systems Engineering Process [2]

13.2.2.1 Process Inputs

The inputs to this process consist of customer requirements in the form of needs, objectives, missions, environments, constraints, and measures of effectiveness. Other inputs include the available technology base, previous program requirements, program decision requirements, and standards and specifications requirements. [2]

13.2.2.2 Requirements Analysis

The systems engineering process begins with *requirements analysis*. During this activity missions and environments are analyzed and functional requirements are identified. Additionally, performance and design constraint requirements are defined and refined. [2]

13.2.2.3 Functional Analysis And Allocation

During the next step, *functional analysis and allocation*, the requirements are decomposed into lower level functions and all performance and constraint requirements are allocated to functional levels. Internal and external functional interfaces are defined and a functional architecture is produced.[2]

13.2.2.4 Design Synthesis

The third activity, *design synthesis*, transforms a functional architecture to a physical architecture. Alternative system elements, concepts, and configuration items are defined and preferred product sets and processes are identified. The internal and external physical interfaces are also defined and refined. This activity is iterated until a system architecture is developed. [2]

13.2.2.5 Process Outputs

The outputs of the systems engineering process include documentation of the alternatives and all decisions. They also include the specifications and baselines and architectures. The architecture development is shown in Figure 13-5.

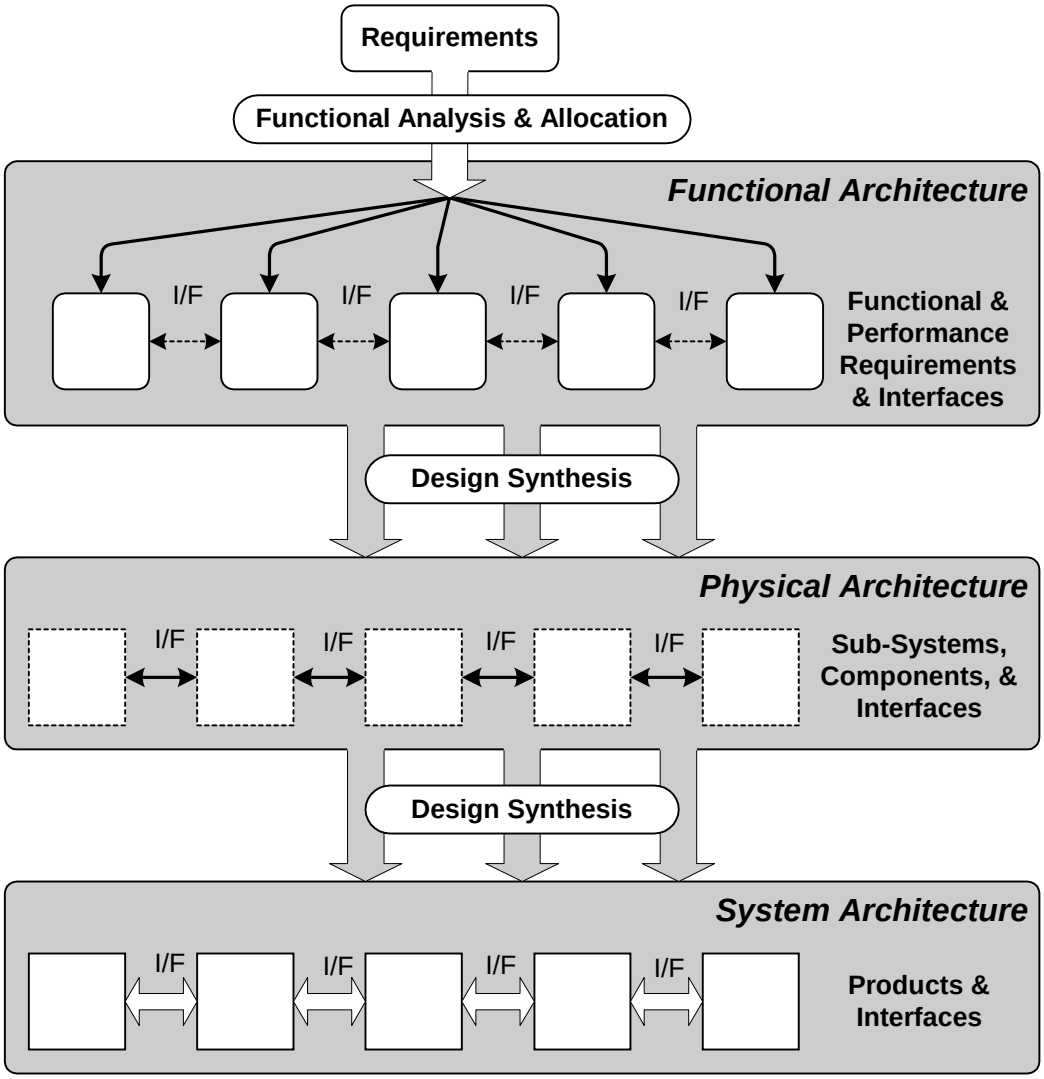

Figure 13-5 Systems Engineering Architecture Development

13.2.2.6 System Analysis and Control

The systems engineering process activities are augmented and controlled by auxiliary *system analysis and control* functions. These functions include activities to measure progress, evaluate and select alternatives, and document all activity and decisions. The following list contains examples of system analysis and control activities.

- Determine and track requirements
- Track decisions and decision rationale.
- Maintain technical baselines.
- Manage and maintain interface definitions.
- Identify and manage risks.
- Control the Configuration.
- Maintain liaison with the customer/user.
- Track cost and schedule.
- Track technical performance.
- Verify requirements are met.
- Review and monitor progress.
- Document the Configuration.

13.2.2.7 Other Activities

The *requirements and design loops* are used to reevaluate the previous work in light of current activities. It is often found that further refinement of requirements, allocations, and designs are necessary and desirable after discovering additional data at the next level of development. The final activity is *verification*. When baselines and architectures are completed, they are compared to the requirements to ensure the solutions fulfill the requirements. All requirements should be verifiable and systems engineering documentation should define the method used for verification of each requirement. [2]

13.2.3 Life Cycle Integration

Life cycle integration is the concurrent consideration of all life cycle needs during the development process. For example, wouldn't it be nice if the manufacturer of your car had given more thought to the location of the oil filter, spark plugs, and other user serviceable components when designing your car's engine? In other words, rather than just producing a system that meets the initial requirements, life cycle integration designs a system or product that considers, as a minimum, the product life cycle functions in the following list. [2]

- Manufacturing/Production
- Verification
- Development
- Deployment
- Operation
- Support
- Disposal
- Training

DoD policy requires integrated product development to be implemented at all levels of acquisition. This is usually accomplished through the use of Integrated Product Teams (IPTs). IPTs are comprised of members from multiple disciplines to ensure considerations from all life cycle phases and functions are considered when developing a product. [2]

13.2.4 Commercial Off-The-Shelf (COTS)

A significant change to military acquisition that has been implemented during the last decade is the use of commercial components and assemblies that are already developed and available for sale in the public marketplace. The idea behind this change is, "If there is a commercial product already built that will satisfy our requirements, let's buy it instead of developing our own." This may require the lowering of standards so that the commercial product will meet the requirements. However, if a realistic appraisal determines that military specifications are not needed in a particular acquisition, there are several *potential* benefits that may be realized. They include: [3]

- Reduced or eliminated development costs.
- Reduced or eliminated development schedule.
- Available skill base.
- Lower acquisition costs.
- Product Improvements paid for by vendor.
- Wide user base to prove the product.
- Industry investment in technology base.
- More component or vendor alternatives.

So what was the reason to buy military components in the first place? First, the military often needs to be able to operate under more severe environmental conditions than the average buyer. Because of the critical nature of the military's mission, drawings, specifications, and source code are often needed to be able to maintain systems as long as the military needs them, not just until the industry decides to move on to something else. There are other factors to consider, including security issues, product changes, safety issues, and compatibility with existing systems.

While it may appear to be a case of "faster and cheaper" COTS components vs. "better" mil spec components, there are a number of risks that must be considered before deciding to acquire COTS instead of developing an item. These are some of the major concerns: [3]

1. Support (maintenance and logistics) may not be responsive enough to meet your requirements.
2. Unforeseen environmental conditions may fall outside the COTS product specifications.
3. There may be incompatibilities with hardware, software, processes, or the operational environment.

4. Verification and validation effort and costs may be higher than anticipated.
5. Integration may be more difficult than estimated.
6. Training costs may be higher than for government-developed systems.
7. Operation and maintenance costs may be higher than for equivalent government components.
8. Vendor viability (technical proficiency, stability, dependency on other sources).
9. Security questions may be unresolved. Is there hidden, malicious code, a "back door," or easily bypassed security checks?
10. Product volatility. Product changes are subject to the vendor's choices and timing.
11. Product quality may be lower than required, impacting reliability, safety, maintainability, and other considerations.

In order to reduce the effects of risks involved with COTS products in an acquisition or development project, the following mitigation techniques should be employed. [3]

1. Thoroughly understand the requirements of the system to be built.
2. Use good systems engineering practices, i.e., understand the functions the COTS software is to perform and the necessary interfaces with the remainder of the system.
3. Gain knowledge about the marketplace and vendors. Know which are viable.
4. Learn about the products, i.e., the functions performed and the required interfaces, to be able to make informed judgments.
5. Understand which things must change the least and which things are likely to change the most.
6. Know all the options.
7. Conduct a make vs. buy vs. rent trade study.
8. Reduce integration and other issues by designing around a major COTS product.
9. Employ industry standards wherever feasible.
10. Establish a robust verification plan and environment.
11. Involve the vendor throughout the life cycle.
12. Get product or vendor certification if possible.
13. Have vendor put source code into "escrow" for future needs.
14. Consider a product line approach.

13.3 Systems Engineering Checklist

This checklist is provided to assist you in understanding the systems engineering issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

13.3.1 Systems Engineering

- ❑ 1. Do you understand the systems engineering process?
- ❑ 2. Are you implementing an optimal systems engineering process?
- ❑ 3. Have you implemented proper and sufficient systems engineering controls and techniques?
- ❑ 4. Are you implementing systems engineering across the whole development life cycle?
- ❑ 5. Is there an experienced and skilled systems engineer directing the systems engineering effort?

- ❑ 6. Is a systems engineering representative providing input to or comments on all product change proposals?
- ❑ 7. Is the systems engineer seeing that all the various development efforts are coordinated and integrated?
- ❑ 8. Do you know what software development life cycle your project will be employing and how it coordinates with the software and project life cycles?
- ❑ 9. Are you considering all phases of the entire life cycle in your requirements, architectures, and designs?
- ❑ 10. Are you implementing an integrated product environment?
- ❑ 11. Have you established integrated (interdisciplinary) product teams?
- ❑ 12. Have you included all the necessary disciplines on the integrated product teams?
- ❑ 13. Are you documenting all studies, decisions, and configurations?
- ❑ 14. Have all internal and external interfaces been defined?
- ❑ 15. Are all your requirements verifiable?
- ❑ 16. Do all your requirements trace to products and vice versa?

13.3.2 COTS [3]

- ❑ 17. Do you conduct make vs. buy vs. rent trade studies instead of just assuming that buy is the right choice?
- ❑ 18. Do you use your requirements as the criteria for your trade studies?
- ❑ 19. Do you consider the full life cycle when deciding whether to make, buy, or rent?
- ❑ 20. Do you know all your options?
- ❑ 21. Are you knowledgeable of the marketplace and the vendors?
- ❑ 22. Does the vendor understand your needs?
- ❑ 23. Are you not lowering your requirements indiscriminately to use COTS?
- ❑ 24. Do you understand the total life cycle costs?
- ❑ 25. Are the product and vendor likely to be around for the lifetime of the system?
- ❑ 26. Have you satisfactorily resolved all security issues?
- ❑ 27. Have you reduced all known risks to an acceptable level?

13.4 References

[1] Sarah A. Sheard, et al, "Systems Engineering Beyond Capability Models," Proceedings of INCOSE, Aug. 2002.

[2] *System Engineering Fundamentals*, 2001, Defense Acquisition University: www.dau.mil/pubs/gdbks/sys_eng_fund.asp

[3] Kohl, Ronald J., "COTS Based Systems: Benefits, Potential Risks and Mitigation Techniques": www.geia.org/etmconf/Workshop/sed/COTS_Issues.pdf

13.5 Resources

Commercial Item Acquisition Report, OSD. Download PDF: www.acq.osd.mil/ar/doc/cotsreport.PDF

Commercial Item Handbook, OSD. Download PDF: www.acq.osd.mil/ar/doc/cihandbook.pdf

CPATS – Systems Engineering, Defense Acquisition Deskbook: http://web1.deskbook.osd.mil/htmlfiles/rlframe/REFLIB_Frame.asp?TOC=/htmlfiles/TOC/019gztoc.asp?sNode=L2-1&Exp=N&Doc=/reflib/daf/019gz/001/019gz001doc.htm&BMK=C1021

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "Opportunities and Complexities of Applying Commercial-Off-the-Shelf Components": www.stsc.hill.af.mil/crosstalk/1998/04/applying.asp
- "An Activity Framework for COTS-Based Systems": www.stsc.hill.af.mil/crosstalk/2000/09/brownsword.html
- "The Double-Edged COTS IT Sword": www.stsc.hill.af.mil/crosstalk/1998/04/publisher.asp
- "Evaluating COTS Using Function Fit Analysis": www.stsc.hill.af.mil/crosstalk/2000/02/holmes.html
- "A Web Repository of Lessons Learned from COTS-Based Software Development1": www.stsc.hill.af.mil/crosstalk/2002/09/rus.html
- "A COTS-Based Replacement Strategy for Aging Avionics Computers": www.stsc.hill.af.mil/crosstalk/2001/12/haldeman.html
- "The Commandments of COTS: Still in Search of the Promised Land": www.stsc.hill.af.mil/crosstalk/1997/05/commandments.asp
- "Lessons Learned From Using COTS Software on Space Systems": www.stsc.hill.af.mil/crosstalk/2001/06/adams.html

DERA Systems Engineering Practices Reference Model: www.incose.org/stc/SEGD12_2.htm

Hitchens, Derek, Systems Thinking, Engineering, & Management site: www.hitchins.co.uk/STEM.html

International Council on Systems Engineering: www.incose.org/

NASA Systems Engineering Handbook:
ldcm.gsfc.nasa.gov/library/ NASA%20Syst%20Eng%20Handbook.pdf

Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapter 7: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

SEI 1998 Software Symposium on COTS. Download slides: www.sei.cmu.edu/cbs/cbs_slides/98symposium/

Sheard, Sara A., et al, "Systems Engineering Standards and Models Compared": www.software.org/pub/ExternalPapers/9804-2.html

Sheard, Sara A., et al, "Systems Engineering Beyond Capability Models": www.software.org/pub/ExternalPapers/SEBeyondCM.doc

System Engineering Fundamentals, 2001, Defense Acquisition University, download at: www.dau.mil/pubs/gdbks/sys_eng_fund.asp

Systems Engineering Guide, Version 1.1, 5 April 1996, ASC/EN -- SMC/SD: http://web1.deskbook.osd.mil/reflib/DAF/073GZ/001/073GZ001DOC.HTM

University of Maryland, COTS Lessons Learned database: http://fc-md.umd.edu/ll/index.asp

Chapter 14

System Integration

CONTENTS

This page intentionally left blank.

Chapter 14

System Integration

"Like a jigsaw puzzle: you have to make the pieces fit without getting out the scissors." – Dr. Karl Maurer – On translating Greek sentences [1]

14.1 Introduction

It may have happened late on Christmas Eve, or it could have been almost any other time. You bought or received something with the *some assembly required* caveat attached. You spent untold time trying to assemble something that was supposed to be easily assembled, but couldn't see how the pieces fit together. Even after humbling yourself to the point of referring to the instruction manual, the puzzle remained a mystery, and was only solved through trial and error, repeated calls to the manufacturer, or the help of a friend who had already gone through the test. Think of the people who had to design the diabolical contraption in the first place, having to find or design each part and make all of them work together.

System integration is the successful putting together of the various components, assemblies, and subsystems of a system and having them work together to perform what the system was intended to do. It follows the coding phase in the development life cycle, as shown in Figure 14-1, and is intertwined with the testing.

Figure 14-1 Integration's Place in the Development Life Cycle

While it may sound like the final assembly of the parts of a system, successful system integration involves almost every aspect of the project and reaches from the very beginning into and through the maintenance phase of a system's life cycle. Figure 14-2 shows the actual integration, where the system comes together, many of the results of successful integration, and several of the activities that are required for successful integration.

Successful system integration results from the proper implementation of project activities shown on the left side of Figure 14-2. The primary requirement and driver is systems engineering (see Chapter 13). When systems engineering is employed throughout the project, successful system integration is one of the primary outcomes. This includes requirements definition, functional analysis, synthesis, trade studies, careful interface definition, true life cycle integration, etc. In addition to the activities associated with systems engineering, correct employment of other activities such as configuration management, design, risk management, and testing are essential to ensuring all the pieces fit together during integration.

Testing goes hand in hand with integration because it is through testing that we determine whether or not the assemblies, subsystems, and systems operate as they should after integrating them. Testing and integration are part of the development process. If there were no testing, the results of system integration would remain an unknown until acceptance testing.

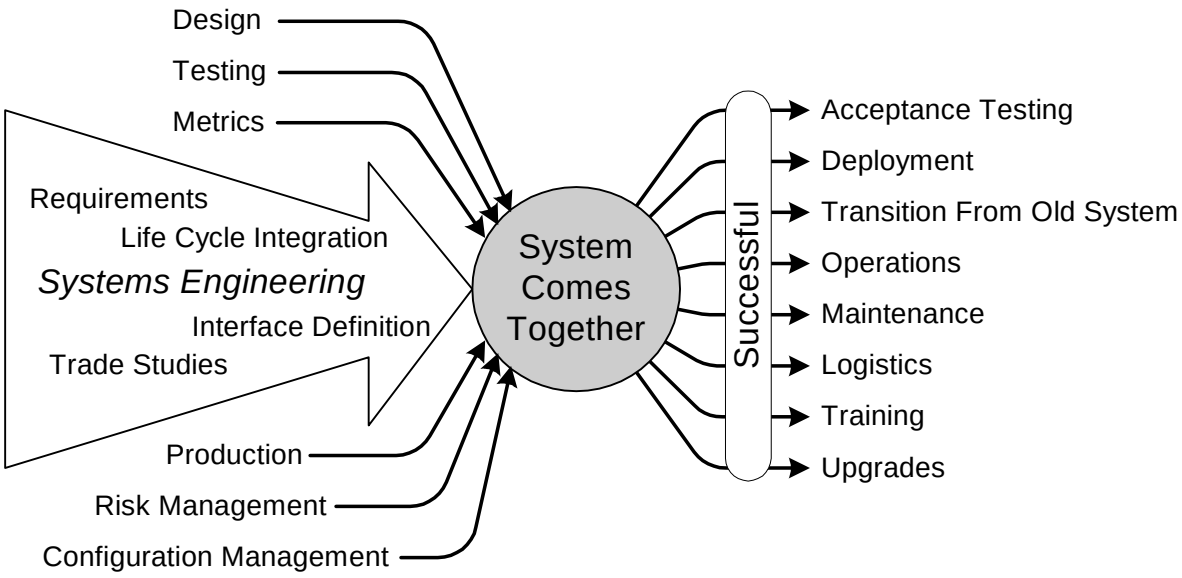

Figure 14-2 System Integration Inputs and Results

14.2 Process Description

As with almost everything else, system integration begins with planning. Because system integration is the logical consequence of systems engineering and other activities, the integration plan is usually a composite of those portions of other plans which pertain to it, as shown in Figure 14-3.

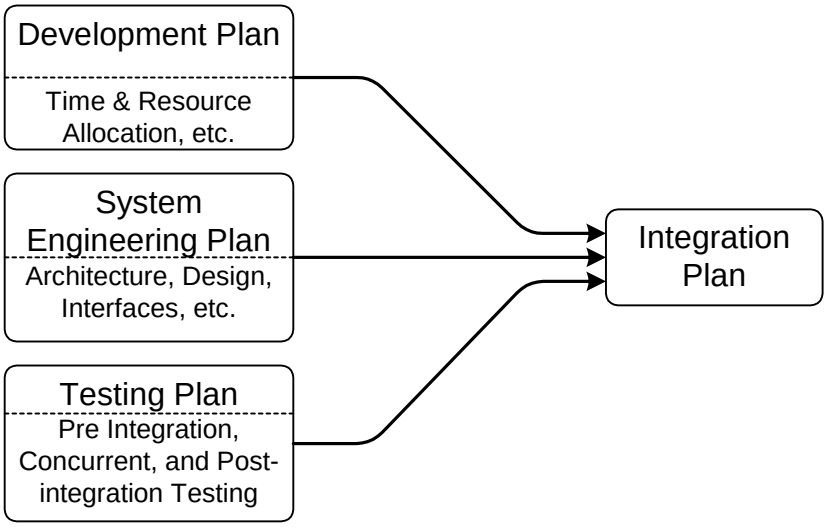

Figure 14-3 System Integration Planning

The integration process was shown in abbreviated form in conjunction with testing in Chapter 12, Figure 12-7. That drawing has been expanded in Figure 14-4 below to depict integration and testing more realistically as a series of integrations and tests. Testing is used to assure developers that the integrated product is properly functional. Integrated modules that fail testing are sent back for debugging and rework.

When tests fail, the test results are analyzed to determine the cause of failure. The components that make up the module being tested are then sent back for debugging and recoding by the developers. If there appears to be a problem with the design, and not with the coding, the module is sent back to design for resolution. When the problem appears to be solved, the integration is repeated and the module is tested again. After rework, all lower level integration and test cycles should be repeated before repeating the higher-level integration. This is part of the regression testing process and detects any new errors that may slip in due to "fixing" other problems.

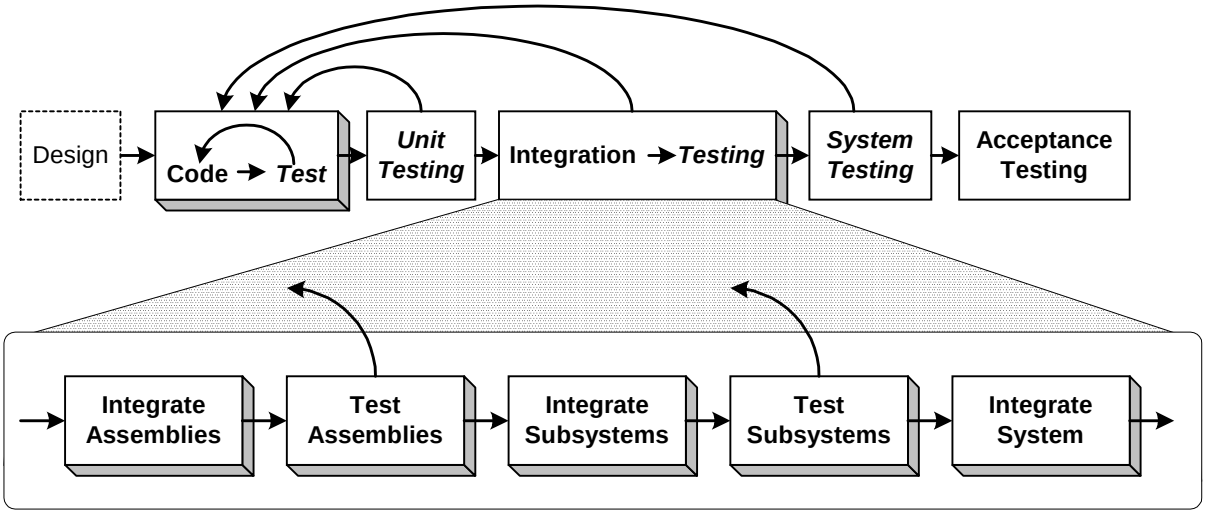

Figure 14-4 System Integration Process

Integration is iterative and progressive, with each level of integration building from and on top of the previous level of integration. This iterative, progressive nature is shown in Figure 14-5. Components are integrated into assemblies, and the assemblies are tested for functionality. Successful testing is followed by the integration of subsystems, which are also tested for correct functionality. Finally, the subsystems are integrated into the complete system, which is then tested for functionality. While three levels of integration are shown in Figures 14-4 and 14-5, it is only representative and in a real project there will probably be additional iterations of integration and testing, depending on the complexity of the system.

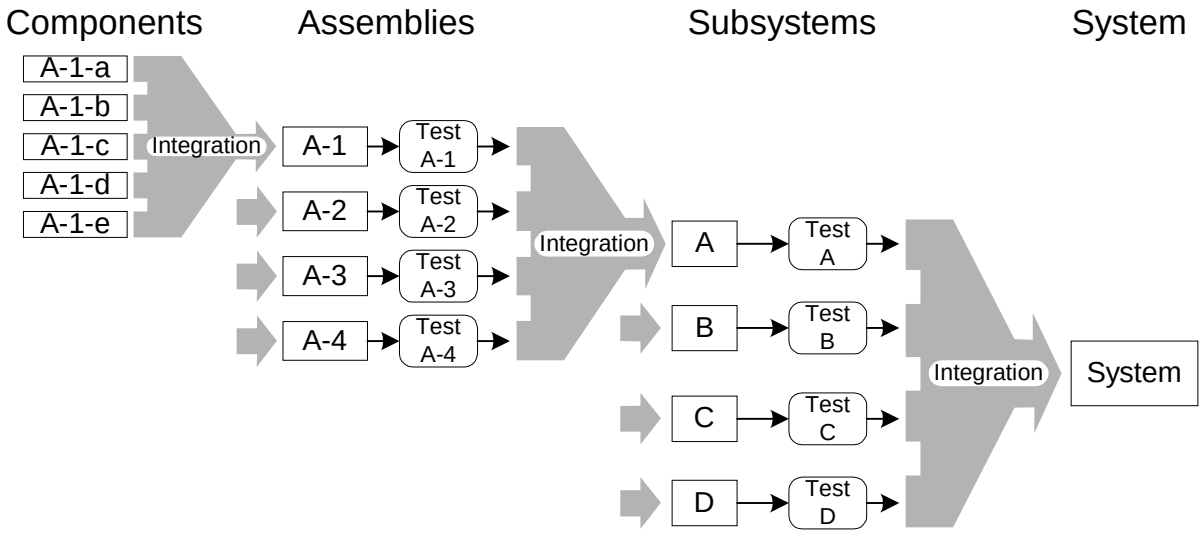

Figure 14-5 Iterative, Progressive Nature of Integration

Integration and testing are part of the development process and are used to ensure all the various pieces work together in performing their higher-level functions.

14.2.1 Interfaces

An absolute essential to any integration effort is complete knowledge of all interfaces. This includes interfaces between components, assemblies, subsystems, and between the system and other systems it will need to work with. This is depicted in Figure 14-6. Defining interfaces and maintaining those definitions is a primary responsibility of systems engineering.

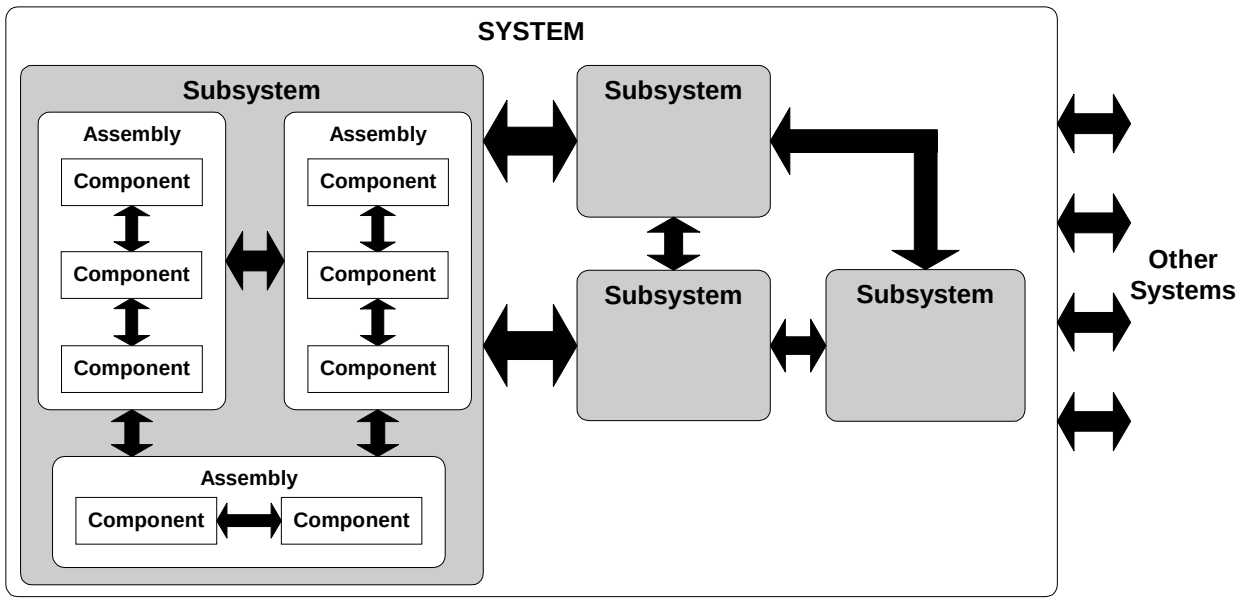

Figure 14-6 Interfaces Between System Parts and Between Systems

14.2.2 Complete System Integration

Most systems consist of both hardware and software. These two are sometimes looked at as complete systems in and of themselves, but they cannot function independently of each other. While they may be called the hardware and software systems, in the system level view they should both be considered as elements of the real, complete system. Development of these two elements may proceed concurrently, with their integration also proceeding concurrently. However, it may be necessary for the hardware to already be in place and operational before the software can be developed, integrated, and tested. Ideally, both elements will be ready for integration into the final system at the same time. If one has to wait on the other, there will likely be problems with schedule, funding, and manpower. The integration of software and hardware elements into a complete system is shown in Figure 14-7.

Figure 14-7 also shows two other system elements: people and support systems. While these other elements may not need to be in place during the development integration, they nonetheless are part of the complete system. For a system to be successfully implemented and used, these other elements must be in place and functioning correctly. The system integration plan must also consider these oft forgotten parts and monitor their establishment. They cannot be left as a follow-on effort. Failure to consider and prepare for all system elements from the beginning will leave the new system crippled or useless.

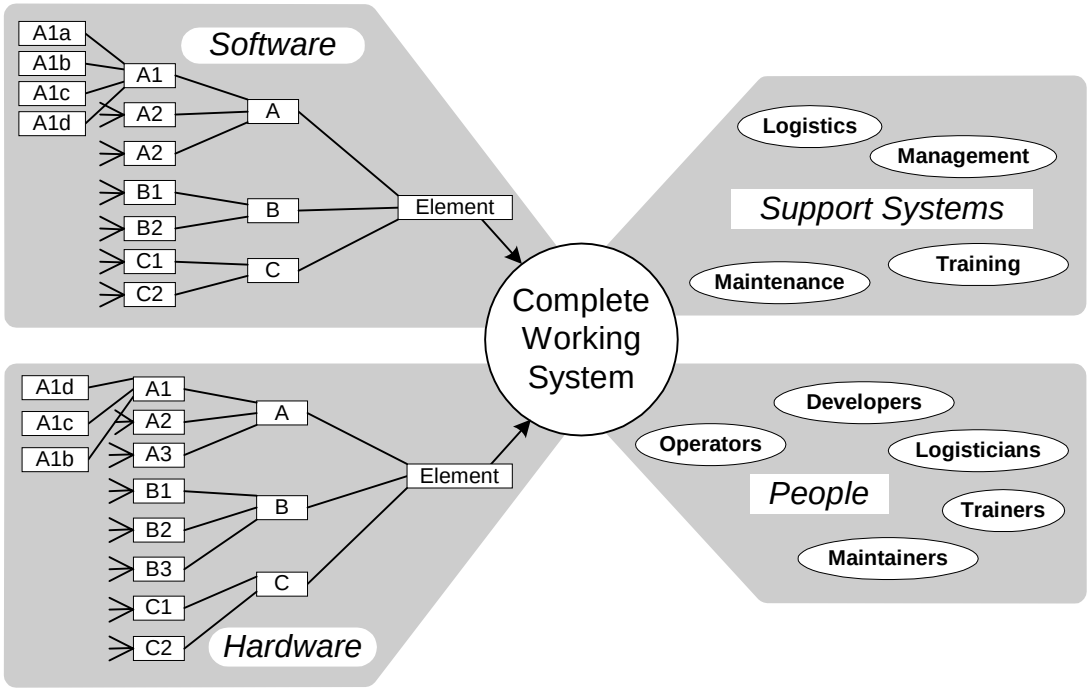

Figure 14-7 System Elements

14.3 System Integration Checklist

This checklist is provided to assist you in understanding the system integration issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

14.3.1 Before Starting

- ❑ 1. Have you implemented systems engineering as an integrated life cycle effort (see Chapter 13)?
- ❑ 2. Do your test plans include and support integration efforts?
- ❑ 3. Does your development plan allocate adequate time and resources for system integration efforts, including rework time?
- ❑ 4. Are the interfaces between components, assemblies, subsystems, and systems defined in adequate detail?
- ❑ 5. Will hardware be available for testing software during integration?
- ❑ 6. Is there a contingency plan if the schedule slips if and the integration schedule is compressed?
- ❑ 7. Are all elements of the system included in the integration plan?
- ❑ 8. Is all documentation current and available for reference?

14.3.2 During Integration

- ❑ 9. Is there an efficient rework cycle in place to fix problems found during integration testing?
- ❑ 10. Are “fixed” modules or components integrated and retested at all levels of integration up to the level where the problem was found?
- ❑ 11. Is the people element (operators, maintainers, logisticians, trainers, etc.) being prepared to work with the system when it is deployed?
- ❑ 12. Is the support systems element (logistics, maintenance, training, etc.) being prepared to support the new system when it is deployed?

- ❑ 13. Are you following an iterative, progressive integration process?
- ❑ 14. Are experienced integrators involved with the integration?
- ❑ 15. Are area/subject matter experts involved with the integration?
- ❑ 16. Is adequate time being allowed for integration, testing, rework, reintegration, and retesting?
- ❑ 17. Are all necessary resources being made available for integration?
- ❑ 18. Is adequate testing being performed on integrated units (assemblies, subsystems, elements, system) to ensure that there are no surprises during acceptance testing?
- ❑ 19. Are you updating documentation during rework?
- ❑ 20. Are integration and system test errors being traced back to requirements and design? And if so, are the requirements and design being updated?

14.4 References

[1] Maurer, Dr. Karl, "Quotes From Greek Class": www.angelfire.com/ga/dracodraconis/greekquotes.html

14.5 Resources

Department of Energy (DOE) *Software Engineering Methodology*, Chapter 8: http://cio.doe.gov/sqse/sem_toc.htm

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "C++ Component Integration Obstacles": www.stsc.hill.af.mil/crosstalk/1997/05/swanson.asp

Guide to Software Engineering Body of Knowledge, especially Appendix D: www.swebok.org

NASA Systems Engineering Handbook: http://ldcm.gsfc.nasa.gov/library/library.htm

Software Engineering Institute: www.sei.cmu.edu

System Engineering Fundamentals, 2001, Defense Acquisition University, download at: www.dau.mil/pubs/gdbks/sys_eng_fund.asp

Systems Engineering Guide, Version 1.1, 5 April 1996, ASC/EN -- SMC/SD: http://web1.deskbook.osd.mil/reflib/DAF/073GZ/001/073GZ001DOC.HTM

Chapter 15

Software Design

CONTENTS

This page intentionally left blank.

Chapter 15

Software Design

"There are very good reasons for everything they do. To the uninitiated some of their little tricks and some of their regulations seem mighty peculiar…" – Eric Frank Russell – "Men, Martians and Machines"

15.1 Introduction

The first digital computer, ENIAC, was constructed in the mid-1940's and became operational in 1945. It weighed over 30 tons, contained 19,000 vacuum tubes and 1500 relays, and used a little under 200 kilowatts of electricity. Its clock cycled at 100 kilohertz and it could multiply 10 digit numbers 384 times a second. [1] In less than the lifetime of a man, ENIAC has evolved into a technology base where computers pervade our existence, being found in appliances and machines affecting virtually every aspect of our lives. This paragraph is being written on a digital computer which can be operated on a human lap, consumes less than 150 watts, and can perform tens of millions of multiplications a second. The physical side of the computing has gone from tubes to transistors to integrated circuits to Very Large Scale Integration (VLSI), and continues unabated in its headlong rush to make everything ever smaller and ever faster.

During this same period the initially experimental craft of programming digital computers has gone through several major revolutions and countless innovations to become a science requiring a lifetime's study to fully comprehend. Any photograph of the current state of computer programming will have become a page in history by the time it is developed. Even the meaning of the sentence you just read will become archaic, if not undecipherable, within a few years because of the convergence of computing and photography. And so it continues.

Computer software began as machine-specific binary code instructions, or ones and zeroes. Thus the first generation of software and programmers were artists as much as anything else, creatively trying to write programs small enough to fit into the limited memory, and fast enough to be useful with the limited processing power. Second generation programming took a step away from ones and zeroes to use mnemonics, special words that represented binary code instructions, to write their programs. Third generation programming incorporated procedural languages in which a single word might be translated into hundreds or thousands of binary instructions. Fourth generation languages attempted to move programming from the realm of telling the computer how to do something, to telling the computer what you want done and letting it write its own procedural program. Fourth generation programming has not been fully successful and software is still generally written in procedural languages with some fourth generation additions and other enhancements. Software development is one of the fastest evolving fields of engineering there are. To cut development costs and time new methods and processes are constantly being created. Software development has become so progressive that its innovations are often applied to other disciplines to help them deal with their own changes.

In spite of all this talk of computer programming, the major work of programming a computer is not programming, or writing software code, but software design. Software is now usually very complex in nature and often consists of multiple programs and resource files. To create good software requires discipline, training, creativity, a lot of work, and good software design processes. This chapter examines the overall software design process and samples some of the design methods used within that process.

15.2 Process Description

Software design is part of the systems development life cycle. Remember that most projects include the development of other system elements in addition to software. Software development rarely has the luxury of proceeding independently of other development, but is constrained to some extent by the other elements and the system as a whole. Software development also has its own life cycle (see Chapter 2) that further constrains the software design effort, determining what methods can be used and how they can be used.

What is presented here as a software design process is a generalized form that will have to be tailored to conform to the various life cycles involved. Additionally, new development methods are evolving to match various types of development with more streamlined and efficient development. While some of these are presented here, the list is incomplete and only representative.

15.2.1 Design Process

The software design phase falls between requirements definition and integration. As shown in Figure 15-1, the inputs to the software design process are the software requirements. The software developed during the design phase is passed on to the integration and test phase. Requirements, integration and test are covered in other areas of this book (Chapters 4, 10, 12, and 14). The design process involves an iterative series of progressively more detailed phases that eventually produce coded software modules ready for integration. Each of these phases is further elaborated below. Remember that there may be more activities or additional iterations of the three shown depending on the development life cycle used. For example, the spiral development model (Section 2.2.1.4) will go through this process two or more times to arrive at the final product.

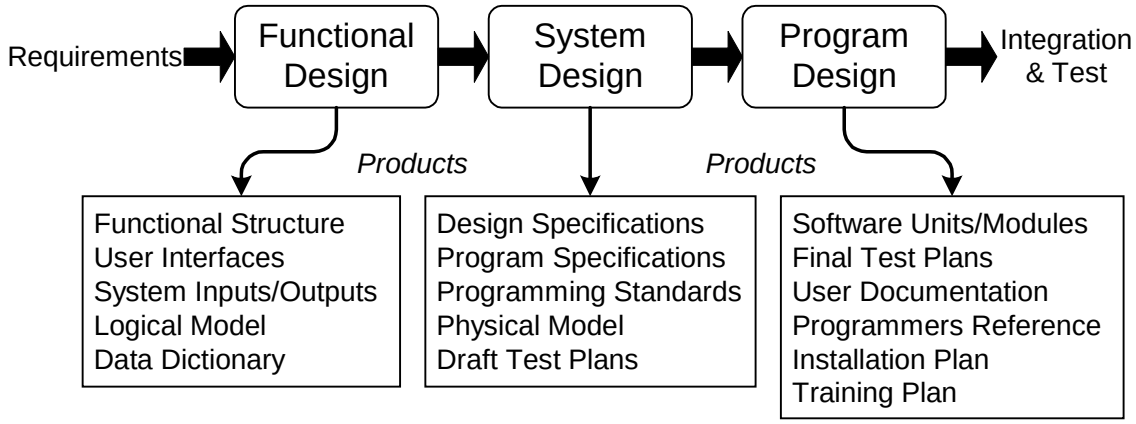

Figure 15-1 Software Design Activities

Note: Functional design and system design may be called by other names, such as preliminary and detailed design.

15.2.1.1 Functional Design [2]

The functional design phase converts the Software Requirements Specification, which tells what the software must do, into a functional design specification, describing how to do it. The functions and structure of the software are defined, including:

- Logical System Flow
- Data Organization
- System Outputs
- System Inputs
- Processing Rules
- Operational Characteristics (User's Viewpoint)

While the focus is on the functional structure of the software, prototyping is often used during this activity to demonstrate the functional design to users and other stakeholders.

The following major activities are performed during functional design:

- Define Software Structure
- Define Content Of System Inputs
- Design User Interfaces
- Define Content Of System Outputs
- Design System Interfaces
- Design System Security Controls
- Build Logical Model
- Build Data Model

- Develop Functional Design
- Conduct Structured Walkthroughs
- Procure Hardware And Software
- Conduct Functional Design Review

The products of the functional design phase include:

- Functional Design Document
- Data Dictionary
- Expanded Requirements Traceability Matrix
- Logical Model
- Design Records
- Hardware And Software Procurement Records

15.2.1.2 System Design [3]

The system design phase converts the user-oriented functional design into technical, computer-oriented, detailed system design specifications. Individual software modules, routines, processes, and data structures are defined, while maintaining the interfaces defined in the functional design phase.

The major activities of this phase include:

- Select System Architecture
- Develop System Design
- Develop Program Specifications
- Develop Conversion Plan
- Develop Integration Test Plan
- Conduct Structured Walkthroughs
- Design Software Module Specifications
- Develop System Test Plan
- Define Programming Standards
- Conduct System Design Review
- Design Physical Model And Database Structure

The products of the system design phase include:

- System Design Document
- Design Specifications
- Physical Model
- Draft Integration Test Plan
- Conversion Plan
- Program Specifications
- Programming Standards
- Expanded Data Dictionary
- Draft System Test Plan
- Expanded Requirements Traceability Matrix

15.2.1.3 Program Design [4]

The function of the program design phase is to produce the actual working software modules specified in the System design phase. The program specifications from the system design phase are used to design and code the individual program modules. Each module must conform to the design documents produced in earlier phases and be thoroughly tested in readiness for integration and testing. Additional activities in this phase include documentation and test planning.

The following activities are included in this phase:

- Write Programs
- Generate Operational Documents
- Develop Training Program
- Plan Transition To Operational Status
- Conduct Unit Testing
- Conduct Structured Walkthroughs
- Establish Programming Environment

The products of this phase include the following:

- Software Units And Modules
- Final Integration Test Plan
- Project Test File
- Users Manual
- Training Plan
- Installation Plan
- Final System Test Plan
- Transition Plan
- Programmers Reference Manual

15.2.2 Design Methods

Originally, almost anything went as far as programming methodology was concerned. Programmers all had their own styles and idiosyncrasies. The problems associated with this *laissez faire* approach began to surface when programmers had to work together on larger, more complex projects. Without a common design method, they had problems with module boundaries, interfaces, data structures, and the list went on. The problems became insurmountable during the maintenance phase when software had to be updated and the original programmer had moved on. It was often easier to build new software than to update the previous version. To help overcome these obstacles, design methodologies were developed and implemented throughout the industry. While the level of success of these methods varies, it is a fact that implementing design methods is far better than not using them.

Design methods are generally used within the framework of the design process discussed in the previous section. The method and/or the process may be modified to combine the two. There is not enough time or room here to make an in-depth study of all the design methods that are available or in use. The three presented here are considered with the briefest regard. They were chosen because they represent an evolution in software development methods. Learn the essential principles of the methods being used on your project. Know their advantages and disadvantages and why they were chosen.

15.2.2.1 Structured Design

Structured design was a major step toward taming the Hydra of spaghetti code programming. It imposed various rules for software design that made it much easier to document, follow, maintain, and in spite of the complaints of some programmers, even easier to design. Two major rules of this method were that (1) programs were to be broken into functions and subroutines, and (2) there was only a single entry point and a single exit point for any function or routine. This imposed an order heretofore unknown in software development and was a major step in making it into a discipline. While other methods have provided even greater order and capacity, most software code is still structured at its lowest level.

15.2.2.2 Object Oriented Design

While structured design had vastly improved the software industry, there were still problems. A change in one part of the program often required thorough searching of the entire program to determine what effects the change had caused in other routines. A simple design change in a single subroutine could ripple throughout the whole program. Code that affected a single decision might be spread throughout the program. Additionally, variables and program structure were accessible to other developers. It wasn't so much a problem of multiple people being able to affect and modify all the code, as it was that those other people *had* to be mindful of what was going on in subroutines throughout the program so they wouldn't inadvertently interfere with them. And there were other issues.

Object Oriented Design (OOD) was developed to overcome many of the problems left unsolved by structured design. It introduced a whole new way of looking at software and currently forms the basis of most new software development. In OOD, a model of the real world is developed to characterize and test the design before having to build it. The model is populated by *objects*, representations of real world objects that have attributes and functions. OOD also implements *encapsulation*, where data and other information are hidden within objects. Other developers can use an object without having to worry about the data inside or how the object operates. One only needs to know what goes into the object, what comes out, and what operations it performs. Another concept implemented under OOD is *inheritance*, where data and objects can be created from previously defined parent data and objects, and have all the attributes of those parents. A third concept, *polymorphism*, allows multiple objects to be modified

through the modification of a single entity. These three concepts greatly enhance developers' abilities to more easily manage multiple objects, reuse and maintain software, and deal with complex software systems.

15.2.2.3 Extreme Programming [5] [6]

A relatively new method of programming is currently making inroads in many small to medium size software development efforts where requirements are vague or rapidly changing. Known as Extreme Programming (XP), it views risk as the basic problem of software development and implements "common sense" approaches to overcome various risks inherent in design. Table 15-1 lists several common risks and their XP solutions.

Table 15-1 How XP Deals With Common Risks

Risk	XP Recommended Mitigation
Schedule slips	Implement short release cycles so that the scope of any slip is limited. Implement higher priority features first so features that slip past the release date are lower value.
User mission misunderstood	The user is an integral part of the team. The specification is continuously refined during development with both developer and user learning.
Mission changes	Release cycles are shortened so there is less change during the development of a single release.
Software rich with unneeded features	Features are implemented in order of their priority (value).
Staff turnover	Programmers are given responsibility to estimate and complete their own work. They receive feedback to improve their estimates. Communications between team members is encouraged to reduced feelings of isolation.
High defect rate	Software is tested from both the programmer's and user's perspectives, function by function, and program feature by program feature.
Project is cancelled	The user chooses the smallest release that makes the most sense mission-wise so there is less to go wrong and software value is the greatest.
Software system goes bad	A comprehensive set of tests is created and maintained, and then run multiple times after every change to ensure a quality baseline.

Extreme programming gets its name by doing good things to extreme levels. Because code reviews are good, code is reviewed all the time. Because testing is good, everyone tests all the time, even the users. Because architecture is important, everyone defines and refines the architecture all the time. If short iterations are good, make them very short, even hours or minutes instead of weeks, months, or years.

Obviously, XP is not applicable to all projects, but it can be successful in the proper setting. And if XP may not be right for a specific project, some of its principles and strategies may be. It also shows us one of the directions software development is heading.

15.2.3 Other Design Considerations

15.2.3.1 Programming Guidelines

The software development guidelines for your project should be documented in a Software Standards and Conventions Document (SSCD). This document defines the following:

1. Overall software development process, including each of the design phases.
2. Programming language quality, style, and standards guidelines.
3. Documentation Standards.
4. Guidelines for use of design tools.
5. Reuse strategies.

6. Software configuration control.
7. Test standards.
8. Review and inspection processes.
9. Metrics to be used.

15.2.3.2 Reuse

Because of the high cost of software development, considerable consideration should be given to designing software that is reusable wherever possible. Designing for reuse begins in the planning phase and must be considered throughout the project to be effective. Proper implementation of reuse techniques and modern design methods, such as OOP, will make maintenance easier and less costly, and can even provide software modules for other projects.

15.2.3.3 Computer Aided Software Engineering (CASE)

CASE tools are software programs that assist the software engineer in designing and documenting software. They are usually dependent on the design methodology employed by the development team, and may even be dependent on the language chosen for implementing the design. Their chief contributions consist of the following advantages:

- Encourage or enforce a formal design process or methodology.
- Document the design in a consistent, formal manner.
- Track the details to help ensure things aren't forgotten.
- Unify the development team in their efforts.

In addition to the above advantages, some CASE tools help in discovering errors, developing tests, tracking requirements, and simulating the software design. Every effort should be made to select an easy-to-use, functional, helpful, and proven tool that not only integrates with your development process, but actually facilitates, assists, and documents that process in as many areas as possible.

15.2.4 Example Design Process

The following example is a process currently being used in the development of an upgrade to an Operational Flight Program (avionics) for an aircraft. The original software already exists so this upgrade will provide additional functionality. The project uses object oriented programming, and is implemented in Ada. There are three design phases, identified as Preliminary, Detailed, and Implementation.

15.2.4.1 PRELIMINARY DESIGN

a. **Review and Allocate Requirements** - Review all new requirements and allocate them to objects, creating new objects as needed.

b. **Create Description of Objects –** Describe each new object at a high level.

c. **Identify Object Associations** – Identify the associations between objects. This is often done by identifying the messages that flow between the objects and using the flows to identify associations.

d. **Allocate Objects to Subsystem –** Allocate all objects to subsystems in the existing architecture.

e. **Identify Object Attributes –** Define the visible attributes (data portion) of each object.

f. **Identify Object Operations –** Define the visible operations of the object.

g. **Develop Static Structure –** Static structure is developed using Class/Object diagrams in a CASE tool.

h. **Develop Scenarios –** High-level scenarios (sequences of events and responses) are developed for significant events.

15.2.4.2 DETAILED DESIGN

a. **Translate Object Diagrams into Ada** – Translate each Class/Object diagram into an Ada package specification.

b. **Refine Ada packages using Program Design Language (PDL)/Ada –** PDL/Ada is a design methodology that combines Ada language control structures with commentary, allowing evaluation of a design before it is coded. PDL/Ada consists of block comments and only those Ada control statements sufficient to describe the program structure. The following components are included:

- **Block Comments** – These describe processing to be performed and are written in English.
- **Program Structure** – When necessary, show program structure with Ada control statements.
- **Data Definitions** – Define data to be used. Include name, description, units, and range.
- **Object Interface Definitions** – Details of object interfaces are defined and documented.

15.2.4.3 CODING

a. **Coding –** Ada packages are coded according to the specifications of the Detailed Design.

b. **Unit Testing –** Program units are tested to ensure they perform correctly against scenarios before being turned over to integration.

15.3 Software Design Checklist

This checklist is provided to assist you in understanding the software design issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

15.3.1 Before Starting

- ❑ 1. Do you have a well-documented software development process?
- ❑ 2. Do you understand what is to be performed and produced in each phase of the design process?
- ❑ 3. Do you have a Software Standards and Conventions Document (SSCD)?
- ❑ 4. Does the SSCD contain direction in those areas listed in Section 15.2.3.1?
- ❑ 5. Are you familiar with the methods, tools, standards, and guidelines in the SSCD?
- ❑ 6. Are applicable and efficient design methods (OOD, etc.) being implemented on your project?
- ❑ 7. Are the developers experienced in the chosen development process and methods?
- ❑ 8. Is software reuse being considered throughout the development effort?
- ❑ 9. Has an analysis of alternatives been completed?
- ❑ 10. Is the selection of architecture and design methods based on system operational characteristics?

15.3.2 During Design

- ❑ 11. Are CASE tools being used to assist and document the design effort?
- ❑ 12. Does your design process include a robust configuration control process?
- ❑ 13. Is the design effort being properly documented? Adequate but not burdensome?
- ❑ 14. Is your team committed to following the design process?
- ❑ 15. Are all design elements traceable to specific requirements?
- ❑ 16. Are all requirements traceable to design elements?
- ❑ 17. Have all software units been identified?
- ❑ 18. Are the characteristics of all data elements identified (type, format, size, units, etc.)?

15.4 References

[1] Weik, Martin H., The ENIAC Story, 1961: http://ftp.arl.mil/~mike/comphist/eniac-story.html

[2] Department of Energy (DOE) *Software Engineering Methodology*, Chapter 5: http://cio.doe.gov/sqse/sem_toc.htm

[3] ibid, Chapter 6.

[4] ibid, Chapter 7.

[5] Beck, Kent, Extreme Programming Explained, Addison-Wesley, 2000.

[6] Mayford Technologies web site: www.mayford.ca

15.5 Resources

CSIRO-Macquarie University, Design tool resources: www.jrcase.mq.edu.au/seweb/designtool/dt.html

Champeaux, Dennis de, et al, *Object-Oriented System Development*, 1993, readable online at: http://gee.cs.oswego.edu/dl/oosdw3/

Department of Energy (DOE) *Software Engineering Methodology*: http://cio.doe.gov/sqse/sem_toc.htm

Guide to Software Engineering Body of Knowledge: www.swebok.org

Little Book of Software Design, Software Program Managers Network, November 1998. Download at: www.spmn.com/products_guidebooks.html

Object Agency , "Comparison of Object-Oriented Development Methodologies": http://www.toa.com/smnn?mcr.html

Program Manager's Guide for Managing Software, 0.6, 29 June 2001, Chapters 8 & 10: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

Software Engineering Body of Knowledge: www.sei.cmu.edu/publications/documents/99.reports/99tr004/99tr004abstract.html

Chapter 16

Sustainment and Product Improvement

CONTENTS

This page intentionally left blank.

Chapter 16

Sustainment and Product Improvement

"My name is Ozymandias, king of kings: Look on my works, ye Mighty, and despair!"
– Percy Bysshe Shelley, "Ozymandias"

16.1 Introduction

In Shelley's poem, the king of kings proclaims his greatness by warning all others to despair after beholding his unequaled excellence. Ironically, it becomes a warning to all who think lofty thoughts of themselves; no matter how great we think we are, or how well we have done our work, there will always be imperfections. And those imperfections lead to a finite lifetime. Anyone who has built or accomplished something will see his work superceded or his record broken if he lives long enough. The statue of Ozymandias had fallen into ruin until it was nothing but two legs of stone and a face half buried in the sand of the surrounding desert. And thus it is with software. There will come a time when its ability to serve will wane. But don't let thoughts of the end keep you from experiencing the excitement and joy of building and accomplishing something in our time.

Eventually, almost any software or system will be upgraded or replaced. From the moment users get their hands on the new system they will try to break it. Is it because of their evil natures? Yes. But we still need to learn to live with it. Besides, all of us have been users at some point. We realize the software would be better if it did this function or did that function better. It takes too long to start up, to find information, or requires more or less input from us than it should. Or there's a new generation of computers that will allow a whole new generation of functionality. Or, and I hesitate to mention it, there are just a few bugs that need to be fixed. It gets harder and harder to convince people that the bugs are "features."

While it is sometimes better to throw out the old system and build a completely new one, we usually find it more advantageous, time and money wise, to upgrade or fix the original software. During the great Year 2000 scare, companies spent billions of dollars upgrading software that was decades old because it was cheaper and took less time than designing, building, testing, and moving to a new system. This is what software sustainment is all about – fixing problems and adding functionality to existing software.

This chapter covers two approaches to building better software. The first, sustainment, has already been introduced. The second is to improve the process we use to build software, making it better and cheaper simply by the way we do things. One deals with improving an existing product. The other approach is to improve the way we build software in the first place. This is not an either or choice. Most software will go through sustainment cycles, and all software organizations need a program of constant process improvement.

16.2 Software Sustainment

Industry often refers to software maintenance. However, the term "software maintenance" is a misnomer. A more correct term would be software sustainment. Why? Because software is not maintained in the same way hardware is maintained. When hardware fails, the repairperson replaces the failed part with an identical but functioning part. When software fails, the software engineer does not replace the offending code with an identical piece of code. The code must be modified to function correctly. Tests must be performed to verify the revised code corrected the problem. This is why we hear quotes that sustainment will account for somewhere between 60 and 90 percent of the total lifetime costs of a software product. Exactly where that number lies today is unknown. It is sufficient to know that it is costly and that it is the longest phase of the software life cycle. The reasons for this costliness are that software will likely be upgraded several times, and it is often difficult to understand what the original developers were doing in any given part of the software so that it can be modified. Great strides have been made to improve the sustainability of software, including:

- Structured programming
- Object oriented programming
- Developing software with sustainment in mind
- Better programming languages
- Better-defined and better-followed development processes
- Computer Aided Software Engineering (CASE) tools
- Better and more consistent documentation

It should be noted that some of these reasons are based on better technology while others are dependent on the cognizance, consideration, and efforts of developers. Implementing these methods should be considered wherever possible to reduce the total cost and improve the quality of software.

Sustainment is often thought of in the context of fixing bugs, but it can be of four different types, depending on the reason or need. While a sustainment effort may be precipitated by a single type of sustainment need, most efforts include two or more sustainment types. The four types are summarized here. [2]

1. **Corrective Sustainment** – diagnosis and correction of program errors after its release.
2. **Perfective Sustainment** – the addition of new capabilities and functionality to existing software.
3. **Adaptive Sustainment** – modification of software to interface with a changing environment.
4. **Preventive Sustainment** – modification of software to improve future maintainability or reliability.

Corrective sustainment requires examination of the existing program code to determine the cause of the error, analysis to determine the best way to correct the error without introducing new errors, and regression testing to validate that the original error has been eliminated without introducing new errors. Perfective and Adaptive sustainment usually involve a complete development effort with requirements, design, coding, and integration and test phases. Preventive sustainment is performed by reverse engineering the existing software and re-engineering (redeveloping) it.

16.2.1 Sustainment Process

The software sustainment process can be very similar to a regular new product development process, depending on the software being upgraded. If the software is relatively modern, developed with current languages and methodologies, and properly documented, it becomes a development project where other programmers have already completed their designs and current programmers are walking through the designs and adding to or correcting them. It's almost as if they are all working on the same effort, just separated in time.

At the other extreme, there are programs which are still being used for which there exists no development documentation. The implementation language has not been used for years and there are few, if any, programmers with working knowledge of the language or system. Why not just build a new system? Because the technology of the entire system is old enough that no one working with it knows how everything works or what is going on behind the scenes. Remember, when we fix things we often introduce new errors into the system. If we're working with a weapon system, we need to know exactly what we're doing. Additionally, some older systems are resource constrained, and newer languages may require more resources (memory, processor speed, etc.) than are available.

Figure 16-1 depicts a generalized software sustainment process. As you see, it's very similar to the standard waterfall development effort. Just as the waterfall life cycle is not used for many current development efforts, the sustainment process may also be modified to better fit the type of project.

Any sustainment effort begins with identifying and gathering requirements. What needs to be fixed? What bugs have been found? What additional features need to be added? This is an exercise that involves both the developers and the users. It will also involve meetings where requirements are presented to make sure everyone agrees on them. They will also likely need to be prioritized, since sustainment funds and time are probably limited.

During the analysis phase, the requirements are validated and feasibility studies are performed. What requirements can reasonably be met within the cost and time of the sustainment effort? This requires familiarity with the existing system. The results of the analysis are reviewed with the users and likely result in modification and reprioritization of requirements. If you can't have everything, what is most important to you?

With the requirements agreed upon, software design begins. Existing software modules are redesigned and new modules are introduced. A new software architecture is produced and documented. Design is usually performed in phases as discussed Chapter 15. It includes a functional and a system design, followed by a programming phase. The difference between sustainment and a new development is that some modules already exist and will need to be modified to operate properly in the new system architecture. The programming phase also includes unit testing.

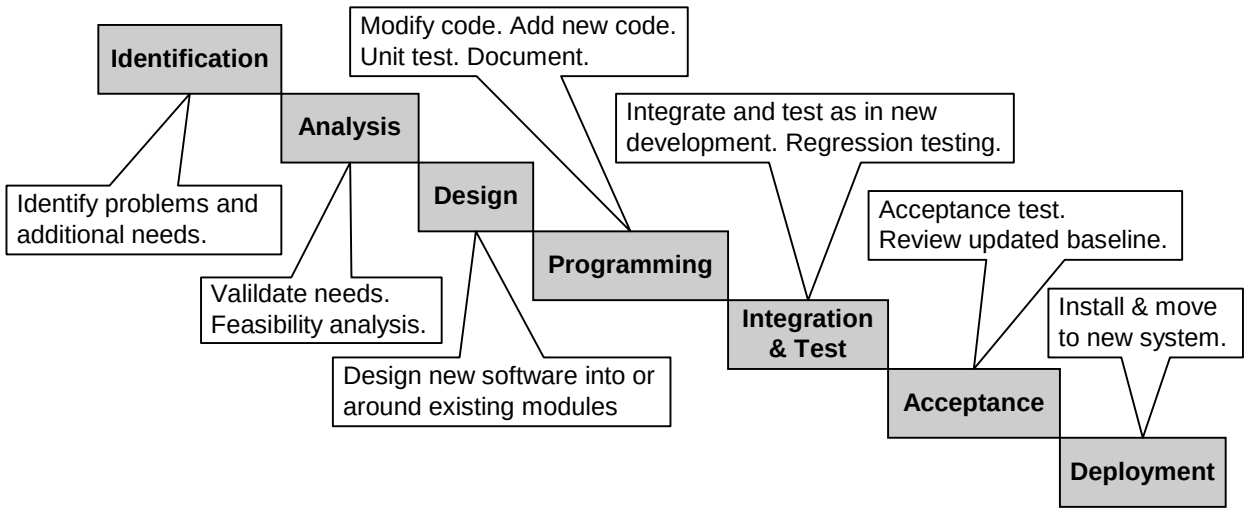

Figure 16-1 Generalized Software Sustainment Process [1]

When the software modules are complete and tested, they are integrated and tested as subsystems and eventually as a complete system. A very critical part of integration and testing is regression testing. Just as the first rule of medicine is to do no harm, the first rule of sustainment is to not destroy existing functionality. Regression testing ensures that original functions are still intact, in addition to the improved functionality sought through the sustainment effort.

When the new system is functioning as required, it is presented to the customer for acceptance testing. When the system is formally accepted, the system baseline is updated to include the new system.

Following acceptance the new system is deployed according to plan. Deployment must follow or be accompanied by adequate training and documentation for operating and maintaining the new system. It must also be performed in such a way as to not interfere unduly with necessary normal operations.

16.2.2 Other Sustainment Considerations

A sustainment plan should be developed for each software system. It should include recommendations for identifying and collecting problems and candidates for improved functionality. It should also include instructions for maintaining proper documentation, source code libraries, and development history for future sustainment efforts. Thought should also be given to maintaining development software (compilers, etc.) for languages that are likely to change or fall into disuse.

In addition to modifying the software, a key sustainment activity is the development of an installation and transition plan to move from the old system to the new. This is a critical activity and should include fallback contingencies if something goes wrong. By the way, few transitions go completely right.

16.3 Product Improvement

Product improvement is the result of consistent, deliberate effort to move the developing organization to a higher level of capability. It requires recognizing and admitting the current state of the organization. It also requires a planned and guided path to excellence, reaching various levels of improvement as intermediate goals along the way. Improvement goes far beyond product quality. It sees product quality as a byproduct of the total development capability of the developers. In other words, if we improve our process, the product will be improved as a natural consequence.

Different groups and professional organizations have from time to time endeavored to establish guidelines and standards for excellence in organizations and business. One attempt to set standards for quality is the International Standards Organization (ISO) 9000 series of standards. Another is the software Capability Maturity Model Integration (CMMI) developed by the Software Engineering Institute at Carnegie Mellon University. While there are other strategies, these two are discussed in this work because of their widespread acceptance and success in the industry.

16.3.1 ISO 9001

The ISO 9000:2000 series seeks to establish standards for business operations, manufacturing, development, and other activities. Of particular interest to us is ISO 9001, the standard in the series that pertains to software development and sustainment. It identifies the minimal requirements for a quality system needed to develop and supply a product. The standard includes various sections pertaining to most aspects of a developing organization. They are:

- General Requirements
- Customer focus
- Organization and communication
- Human resources and training
- Planning of product realization
- Purchasing
- Planning of monitoring and measurement
- Analysis of data
- Documentation and Records
- Quality policy
- Management review
- Infrastructure
- Customer-related processes
- Operations
- Monitoring and measurement
- Continual improvement
- Management commitment
- Quality system planning
- Provision of resources
- Work environment
- Design control
- Measuring and monitoring equipment
- Control of nonconforming product

If an organization meets the minimum standards for these sections it can be certified as complying. ISO 9001 allows organizations to measure their processes against an industry gauge to see where they have strengths and weaknesses. If properly used, the standard shows companies what they need to work on to improve their efficiency, quality, and capacity. It provides a set of goals to which can be tailored for their particular industry and organization. Ideally, it will improve their standing in the market and improve the overall capability of the industry to produce better products, in shorter time, and at better prices.

TickIT is the British and Swedish extension to ISO 9001. It focuses on applying ISO 9001 to software development. More can be found regarding TickIT at www.tickit.org.

While ISO 9001 can be an indicator of competence, it does not guarantee it. If an organization focuses primarily on achieving the standards as a paper exercise, but does not incorporate the intent into the mindset and activity of the organization, the true benefit of the standard will not materialize. This problem can be alleviated by following an improvement strategy that emphasizes continuous improvement rather than meeting a standard. This is where the CMM comes in.

16.3.2 Capability Maturity Model Integration (CMMI)

Originally, there were several different versions of capability maturity models: one for software, one for system engineering, and one for software acquisition. Recently, these separate models have been integrated into a single model, the CMMI. Two different representations are available for the CMMI, a continuous representation (previ-

ously used by the System Engineering CMM) and a staged representation previously used by both the Software and Software Acquisition CMMs). The staged representation shows progress as a series of five levels. Each of these levels is described by certain attributes characterizing its level of competency. Each level is associated with process areas, and each process area is described in terms of common practices that support that level's goals. These levels, descriptions, and process areas are shown in Table 16-1.

Table 16-1 Key Process Areas in the CMM

Level	Description	Key Process Areas
5 – Optimizing	The focus is on continually improving process performance through both incremental and innovative technological improvements.	Organizational Innovation and Deployment Causal Analysis and Resolution
4 – Quantitatively Managed	Quantitative objectives for quality and process performance are established and used as criteria in managing processes. Quantitative objectives are based on the needs of the customer, end users, organization, and process implementers. Quality and process performance are understood in statistical terms and are managed throughout the life of the processes.	Organizational Process Performance Quantitative Project Management
3 – Defined	Processes are well characterized and understood, and are described in standards, procedures, tools, and methods. Projects tailor organizational standards, procedures, and methods for use by the project.	Requirements Development Technical Solution Product Integration Verification Validation Organizational Process Focus Organizational Process Definition Organizational Training Integrated Project Management for IPPD Risk Management Integrated Teaming Integrated Supplier Management Decision Analysis and Resolution Organizational Environment for Integration
2 – Managed	The projects of the organization have ensured that requirements are managed and that processes are planned, performed, measured, and controlled.	Requirements Management Project planning Project Monitoring and Control Supplier Agreement Management Measurement and Analysis Process and Product Quality Assurance Configuration Management
1 – Initial	Processes are characterized as ad hoc, occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics.	–

The level of capability is shown in the first column. It begins at the bottom of the table with the initial level of capability. This is characterized by ad hoc processes. Each new effort is treated differently and processes are seldom repeated. Capability matures through several levels until it reaches the top, or optimizing, level. Rather than being characterized by achievement of a static state, an organization at this level is in a dynamic state of continuous process improvement. This is where CMM differs from ISO 9000. Instead of reaching a plateau or mountain top and

planting a flag, the CMM achiever attains the capability to climb mountains of specific heights and difficulty, and continues to climb that type of mountain while continuously striving to reach higher levels.

An organization begins process improvement by undergoing an evaluation of its current capability. There will usually be a disparity among the capabilities of different areas. Once the level of capability has been ascertained, planning and effort are directed toward improving processes in those deficient areas to move the organization to the next level. Studies have shown that it takes two to three years to advance to a higher level. Since levels are determined by an organization's ongoing activity, those who do not continuously strive for improvement will not only stop advancing, they will regress in their capability level.

Both ISO 9001and CMMI are far too complex to present more than the briefest of overviews here. Study some of the recommended resources at the end of the chapter to gain a fuller understanding of what they are and, more importantly, what they can do for you. If your organization is not implementing an improvement strategy, such as CMMI, it is highly recommended that the advantages and increased capacity of such an effort be considered and evaluated. In today's constantly changing world, no one has the luxury of staying on one place very long.

16.4 Sustainment and Product Improvement Checklist

This checklist is provided to assist you in understanding the sustainment and product improvement issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

16.4.1 Sustainment

- ❑ 1. Is all your software developed with a goal to facilitate its future sustainment?
- ❑ 2. Do you understand the four types of sustainment and their purposes?
- ❑ 3. Do you understand the place and purpose of the sustainment phase in the software life cycle?
- ❑ 4. Do you understand your sustainment process?
- ❑ 5. Is there a sustainment plan?
- ❑ 6. Is there a process in place to gather problem reports and upgrade requests for the software?
- ❑ 7. Does the plan provide for reviewing, evaluating, and prioritizing upgrade requests?
- ❑ 8. Are all sustainment activity steps included in the plan?
- ❑ 9. Is there a transition plan to move to the upgraded system?
- ❑ 10. Have all activities been planned and organized to keep interference and downtime to the operating system to a minimum?
- ❑ 11. Does the plan call for running critical systems redundantly during testing and installation?
- ❑ 12. Do the deliveries include source code, documentation, and all else that is needed in addition to the software itself to continue maintaining the software?
- ❑ 13. Are all products under configuration control?

16.4.2 Product Improvement

- ❑ 14. Is your organization following a product improvement strategy?
- ❑ 15. Do you know where your organization is at, capability wise, relative to ISO 9001, CMMI or your chosen improvement standard?
- ❑ 16. Do you have a plan for achieving higher levels of capability?
- ❑ 17. Do your product improvement efforts emphasize improving processes to achieve product improvement?
- ❑ 18. Are your development processes documented?

- ❑ 19. Are your development processes consistent and repeated?
- ❑ 20. Does the leadership of your organization support continuous process improvement?
- ❑ 21. Is your organization committed to achieving a state of continuous improvement vs. a certificate of compliance with standards?

16.5 References

[1] Department of Energy (DOE) *Software Engineering Methodology,* Chapter 10: http://cio.doe.gov/sqse/sem_toc.htm

[2] *Program Manager's Guide for Managing Software*, 0.6, 29 June 2001, Chapter 12: www.geia.org/sstc/G47/SWMgmtGuide%20Rev%200.4.doc

16.6 Resources

Caputo, Kim, *CMM Implementation Guide*, Addison Wesley Longman, Inc., 1998.

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "Confusing Process and Product: Why the Quality is not There Yet": www.stsc.hill.af.mil/crosstalk/1999/07/cook.asp
- "It's Time for ISO 9000": www.stsc.hill.af.mil/crosstalk/1994/03/xt94d03i.asp
- "Coding Cowboys and Software Processes": www.stsc.hill.af.mil/crosstalk/1997/08/processes.asp
- "Improvement Stages": www.stsc.hill.af.mil/crosstalk/1998/10/cusick.asp
- "The Process King vs. the Cowboys": www.stsc.hill.af.mil/crosstalk/1997/08/backtalk.asp
- "Using the CMM Effectively": www.stsc.hill.af.mil/crosstalk/1995/10/usingcmm.asp
- "If You Get Straight A's, You Must Be Intelligent - Respecting the Intent of the Capability Maturity Model": www.stsc.hill.af.mil/crosstalk/1998/02/respecting.asp
- "Software Process Proverbs": www.stsc.hill.af.mil/crosstalk/1997/01/proverbs.asp
- "Preparing for a CMM Appraisal": www.stsc.hill.af.mil/crosstalk/1996/08/preparin.asp
- "Ten Things Your Mother Never Told You About the Capability Maturity Model": www.stsc.hill.af.mil/crosstalk/1998/09/kulpa.asp

International Standards Organization, ISO 90001, www.iso.ch (also available from http://qualitypress.asq.org)

Jeffries, Ron, "Extreme Programming and the Capability Maturity Model": www.xprogramming.com/xpmag/xp_and_cmm.htm

Paulk, Mark C., "Effective CMM-Based Process Improvement": www.sei.cmu.edu/publications/articles/effective -spi.html

Randall's Practical Resources Online, ISO 9000 and Quality Related Topics: http://home.earthlink.net/~rpr-online/Contents.htm#ISO9000

Software Engineering Institute, CMMI and resources: www.sei.cmu.edu/cmmi/

Software Insight Tool, Checklists for acquisition and risk mitigation, etc.: www.sed.monmouth.army.mil/sit/sitstart.htm

TickIT information, www.tickit.org

University of Massachusetts Dartmouth, Software Process Resource Collection: http://www2.umassd.edu/SWPI/

This page intentionally left blank.

Chapter 17

Acquisition Environment and Regulations

CONTENTS

This page intentionally left blank.

Chapter 17

Acquisition Environment and Regulations

"And so we continue our daily battle against entropy, wondering if in our fight to bring order, we ultimately add to the chaos." – Roald Peterson

Prefatory note: This chapter summarizes the acquisition environment that existed in the first half of 2002. Since that time the acquisition regulations have been declared out-of-date, but new regulations have not yet been released. It is expected that this chapter will be updated following the release of the new regulations.

17.1 Introduction

I went to Paris on a business trip once. After flying through the night, getting through customs, finding my hotel, and taking a nap, I went downtown to see the some sights before it got dark. After walking around the Arch du Triumph I looked for a place to eat supper. I was in a strange place, in a very different time zone, surrounded by a language I didn't understand, and hungry. I didn't need an opportunity to decipher what was to me an alien menu offering unknown foods. I had already been through that on several occasions in South America. I spotted a McDonalds restaurant down a side street and ate there with gusto. Why? I was familiar with the products and could reasonably understand the menu. I knew what I would get and how the system worked.

Regulations are too often viewed as just more obstacles to getting things done, and at times that has been the case. The purpose of regulations is to provide a standardized, formal path for doing things. Ideally, they should take into account the common pitfalls and steer people around them. They should provide a common framework so everyone involved in the process knows what to expect. They also allow practitioners to invoke higher authority to get through roadblocks and obtain cooperation.

This chapter is a condensation of the material in Chapters 3 and 4 of the *Guidelines for the Successful Acquisition and Management (GSAM) of Software Intensive Systems*, Version 3.0, May 2000. For a more in-depth examination of the acquisition environment, see these two chapters.

17.2 Statutory Framework

Acquisition has gone through many iterations of reform, and will likely continue to do so. We live in a time of great change; change in world events, administrations, economies, technology, social orders, and knowledge. If the acquisition environment is to remain relevant it must also change. The chief goals of acquisition reform are:

- Cost savings
- Reduced cycle times
- Program stability
- Technology insertion

17.2.1 Major Acquisition Reform

The *National Performance Review (NPR)*, completed in 1993, recommended a series of steps to create a more responsive, effective, and efficient acquisition environment. Several acts were legislated to implement its recommendations, chief of which was the *Government Performance and Results Act (GPRA)*. The NPR and GPRA are shown in Figure 17-1 with their summarized recommendations and requirements.

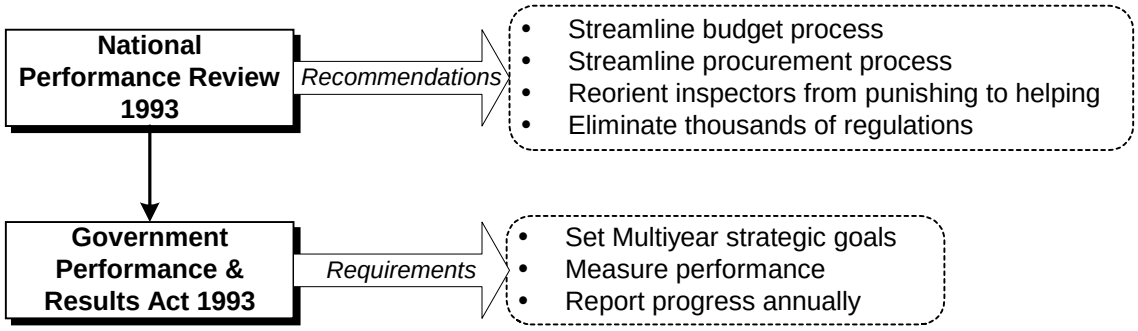

Figure 17-1 NPR Recommendations and GPRA Requirements

The Government Accounting Office's (GAO) *Executive Guide: Effectively Implementing the Government Performances and Results Act* identifies a set of key steps and practices to implement reforms consistent with the GPRA. They are:

A. Define Mission & Desired Outcomes

1. Involve stakeholders.
2. Assess environment.
3. Align activities, core processes, and resources.

B. Measure Performance

4. Produce measures at each organizational level that:
   - Demonstrate results
   - Are limited to vital measures
   - Respond to multiple priorities
   - Link to responsible programs
5. Collect data.

C. Use Performance Information

6. Identify performance gaps.
7. Report information.
8. Use information.

D. Reinforce GPRA Implementation

9. Devolve decision-making accountability.
10. Create incentives.
11. Build expertise.
12. Integrate management reforms.

The Department of Defense (DoD) complies with the GPRA through its Planning, Programming, and Budgeting System (PPBS). Figure 17-2 illustrates the PPBS process along with documents and plans produced and used as by the process.

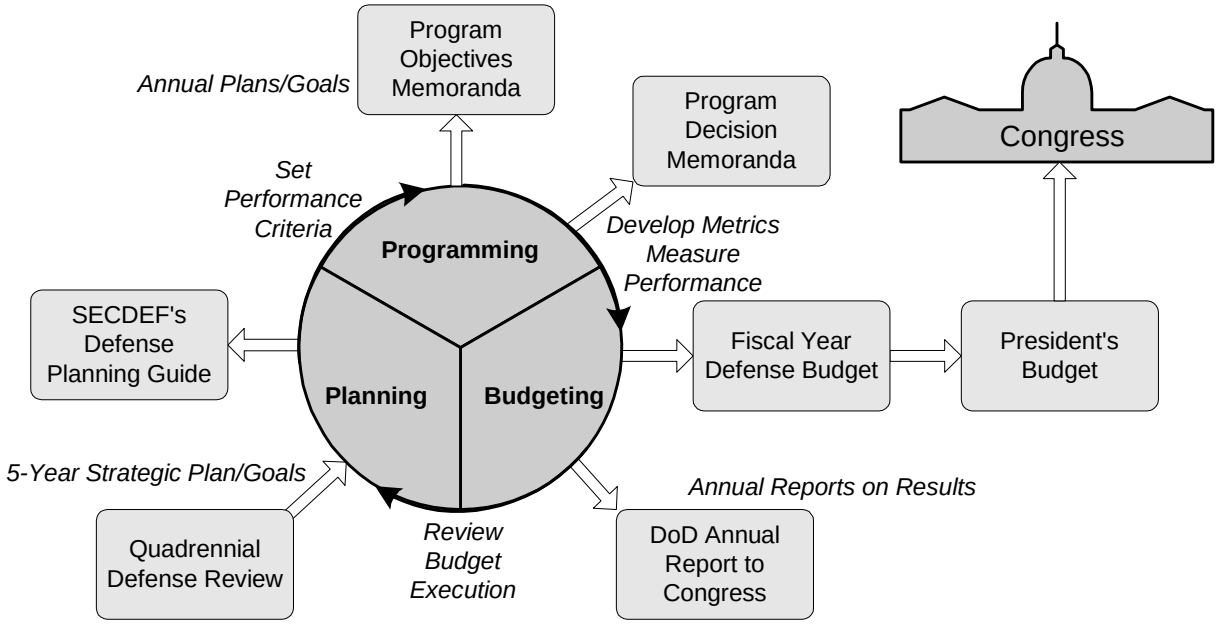

Figure 17-2 DoD Compliance With GPRA Through the PPBS Process

17.2.2 Federal Acquisition Streamlining Act (FASA)

Another acquisition reform act, the Federal Acquisition Streamlining Act (FASA) was enacted in 1994. It removed many rigid acquisition regulations and allowed DoD to implement management best practices. FASA reform provisions pertaining to software system acquisitions include:

- Market research
- Quality and non-price evaluation factors
- Contractor past performance
- FACNET (An automated, electronic list of what the government wants to purchase)
- Commercial buying practices for COTS
- Indefinite Delivery Indefinite Quantity (IDIQ)
- Performance based payments
- Preference for Commercial Off the Shelf (COTS) and Non-Development Items (NDI)

FASA implements the results-oriented, performance-based management requirements of the GPRA. At the end of each fiscal year each major defense acquisition program is evaluated to determine whether it has 90% or more of its cost, schedule, and performance goals, compared to Acquisition Program Baseline (APB) thresholds. If the threshold is missed, a review is required to determine whether a program breech is needed and to recommend suitable action. In addition to tracking acquisition goals, DoD is required to implement an incentive system for acquisition personnel

FASA also requires DoD to report annually on technology insertion, or how it converts emerging technology into operational capability. DoD reduces the time required for technology insertion by:

- Using commercially available technologies
- Encouraging tradeoffs between cost, schedule, and performance at various stages in development
- Expanding the use of Advanced Concept Technology Demonstrations (ACTD).

There have been several other acquisition reform laws but this will suffice to give us an introduction to what they are and what they require. For a more complete treatment, see the GSAM, chapter 3.

17.2.3 Clinger-Cohen Act

The Clinger-Cohen Act of 1996, also called the Cohen Act, the CCA, or CIO Act, was enacted to improve acquisition and management practices pertaining to information resources. Prior acquisitions took so long that systems were often obsolete when fielded. The Clinger-Cohen Act reduced the excessive documentation required for the approval process and shortened the acquisition cycle, leading to the timelier fielding of information systems. The specific requirements of the act are: [1]

- Designation of a Chief Information Officer (CIO) to establish information technology standards and oversee technology spending.
- Decentralize procurement authority.
- Implement capital planning and investment control by making decisions based on measurable criteria related to Return-on-Investment, alternative solutions, shared benefits and costs, and verifiable progress.
- Implement performance and results-oriented management by establishing strategic goals and monitoring progress toward those goals.
- Provide accountability by ensuring information systems provide timely, reliable, and consistent performance data.
- Establish and follow standards and guidelines for information systems efficiency, security, and privacy.
- Ensure that the information technology acquisition process is clear, simplified, and understandable.
- Give procurement protest authority to the US Comptroller General and the General Accounting Office (GAO).
- Improve processes, rather than simply automating existing business functions or replacing old technology.
- Establish a DoD-wide information technical architecture and integrate new systems into that architecture to avoid fragmented, isolated systems.

17.3 Applicable Regulations

The following regulations apply to software acquisitions.

17.3.1 Federal Acquisition Regulation (FAR) & DoD FAR Supplement (DFARS)

The Federal Acquisition Regulation establishes uniform policies and procedures for the acquisition of supplies and services by all executive agencies. It is the highest-ranking statutory document and takes precedence over the DFARS and DoD 5000.1 and DoD 5000.2-R. The FAR regulates procurement planning and Request For Proposal (RFP) development. The DFARS adds DoD unique information and clarification to the FAR. The FAR and DFARS include the following aspects: [2]

- Provides a statement of principles for the federal acquisition system. [FAR 1.102]
- Provides definitions of words and terms used in contracts. [FAR Part 2]
- Prescribes the policy and procedures that are to be used to promote and provide for full and open competition. [FAR Part 6]
- Prescribes policies and procedures for --

  (a) Developing acquisition plans;

  (b) Determining whether to use commercial or Government resources for acquisition of supplies or services;

  (c) Deciding whether it is more economical to lease equipment rather than purchase it; and

  (d) Determining whether functions are inherently governmental. [FAR Part 7]

- Describes various contract types and when they are appropriate. [FAR Part 16]
- Describes the standard acquisition package format and identifying the appropriate content of each section. [FAR 14.201-1]
- Describes the need to disclose beforehand the evaluation criteria to be used for contract award. [FAR 15.304]
- Describes acquisition policies and procedures for use in acquiring major systems [FAR Part 34]
- Prescribes acquisition policies and procedures for use in acquiring information technology [FAR Part 39]
- Limits contracts to a maximum of five years. [FAR 17.1]
- Clearly explains data rights and what must be done to acquire the data rights deemed necessary. [FAR 27.4]

17.3.2 DoD 5000 Regulations Framework

(THIS SECTION TO BE UPDATED UPON RELEASE OF REVISED DoD 5000 SERIES IN SPRING 2003. Check current status of 5000 series at http://dod5000.dau.mil)

The DoD has updated its acquisition policies to comply with the acquisition reform legislation. These police updates are embodied in *DoD 5000.1, Defense Acquisition Directive* (1996), and the associated regulation DoD 5000.2-R, Mandatory Procedures for Major Defense Acquisition Programs (MDAPS) and Major Automated Information Systems (MAIS) (1998). While 5000.1 provides guidance for all DoD acquisition programs, 5000.2-R applies specifically to major programs, allowing decentralization of acquisition and more autonomy for the Component Acquisition Executives to manage their programs.

17.3.2.1 Milestone Decision Authority (MDA)

MDAPs are subject to Milestone Decision Authority (MDA) reviews by the Defense Acquisition Board (DAB). However, the Program Manager (PM) is in charge of the program and the Integrated Product Teams (IPTs) are empowered to help the PM resolve issues before DAB reviews. Where some programs previously had to prepare two quality programs, two test programs, two configuration control programs, etc., the combining of all programs under DoD 5000 reduces the work to fulfilling one set of acquisition requirements.

17.3.2.2 Software-Intensive Systems

DoD 5000.2-R requires that all software development projects be managed and engineered using commercial best practices and processes to reduce cost, schedule, and performance risks. Software-intensive systems must be developed with sound systems engineering principles, including:

- **Architecture** – Implement software systems architectures that support open system concepts, exploit COTS computer products, and provide for incremental improvements based on modular, reusable, extensible software.
- **Reuse** – Identify and exploit software reuse opportunities before beginning a new software development.
- **Programming Languages** – Select languages based on systems and software engineering factors that influence overall life cycle costs, risks, and potential for interoperability.
- **Standard Data** – Use DoD standard data.
- **Successful Contractors** – Select contractors with:
  - Domain experience with comparable software systems.
  - Successful past performance record.
  - Demonstrable software development capability and a mature process.
- **Measurement** – Select contractors with a mature measurement process for planning, tracking, assessing, and improving the development process and products.
- **Risk Measurement** – Assess information system operational risks.

- **Year 2000** – Ensure all software is Year 2000 compliant.
- **Information Security** – Manage and engineer system to reduce security risks, including timely accreditation.
- **C4I Support Plan** – Prepare a support plan for all Command, Control, Communications, Computers, and Intelligence (C4I) systems and all weapon systems that interface with C4I systems, to include:
  - System description.
  - Employment concept.
  - Operational support requirements, including C4I, testing, and training.
  - Interoperability and connectivity characteristics.
  - Management and scheduling.

17.3.2.3 Results-Oriented Acquisition Management

DoD 5000.2-R emphasizes the determination of producibility early in the development cycle. It states that producibility is key to managing risk, and that production should not be approved until the design has been stabilized, the development processes have been proven, and facilities, equipment, and people are in place. Mission Need Statements (MNS) must be linked to the mission described in the DoD Strategic Plan (Quadrennial Defense Review). In addition to linking programs to strategic goals, it emphasizes interrelationships between defining requirements, managing system development, and making funding decisions, with the objective of translating user needs into products that are affordable.

DoD 5000.1 encourages the use of nontraditional acquisition methods where those methods provide advantages over traditional methods, such as lower costs, shorter development cycles, and more rapid fielding of proven technologies. Example of nontraditional methods include modeling, simulation, Advanced Concept Technology Demonstrations, rapid prototyping, evolutionary and incremental acquisition, flexible technology insertion, modular contracting, innovative practices, and Cost As an Independent Variable (CAIV). Use of IPTs removes barriers between different organizations and disciplines and enables integrated solutions to development problems.

17.3.3 DoD Directive 8000.1

DoD Directive 8000.1, Management of DoD Information Resources and Information Technology, 27 February 2002, establishes policies for DoD information resources management (IRM), including information technology (IT), and delineates authorities, duties, and responsibilities for DoD IRM activities. Of particular interest to software acquisition and development programs are sections 4.4 through 4.8. These sections address the following topics:

- The need for accurate and consistent information for decision-makers so they can effectively execute the DoD mission.
- How integrated analysis, planning, budgeting, and evaluation processes must be used to strengthen the quality of decisions about using IT to meet mission needs.
- Analysis that must be accomplished before applying information technology solutions.
- Areas that must be covered in acquisition strategies.
- Risk reduction areas that should be implemented.

17.3.4 DOD Directive 8100.1

DoD Directive 8100.1, Global Information Grid (GIG) Overarching Policy, 19 Sep 2002, establishes policy and assigns responsibilities for GIG configuration management, architecture, and the relationships with the Intelligence Community (IC) and defense intelligence components. Of particular interest to software acquisition and development programs is section 5.7 which requires DoD Components to:

- Populate and maintain their portion of the GIG asset inventory.

- Ensure that the DoD Component architectures are developed and maintained consistent with the GIG architecture.
- Require the use of GIG common computing and communications assets within their functional areas and within their Component.
- Ensure all Component-leased, -owned, -operated, or -managed GIG systems, services, upgrades, or expansions to existing systems or services are acquired or procured in compliance with the current version of the Global Information Grid Architecture.

17.3.5 DoD Directive 8500.1

DoD Directive 8500.1, Information Assurance, 24 Oct 2002, establishes policy and assigns responsibilities to achieve Department of Defense (DoD) information assurance (IA) through a defense-in-depth approach that integrates the capabilities of personnel, operations, and technology, and supports the evolution to network centric warfare. Of particular interest to software acquisition and development programs are Section 4, Policy, and portions of Section 5, Responsibilities. Some excerpts from these sections are:

- Information assurance requirements shall be identified and included in the design, acquisition, installation, operation, upgrade, or replacement of all DoD information systems.
- All IA or IA-enabled IT hardware, firmware, and software components or products incorporated into DoD information systems must comply with the evaluation and validation requirements of National Security Telecommunications and Information Systems Security Policy Number 11.

17.4 Best Practices Initiatives

DoD is reengineering its acquisition processes to provide the warfighter with the best-value goods and services. Reform initiatives are being pursued in the following five categories:

1. Research, development, and test restructuring.
2. Sustainment restructuring.
3. Increased acquisition workforce education and training.
4. Integrated, paperless operations.
5. Future focus areas (i.e., a price-based acquisition and full integration of test and evaluation activities into the acquisition process).

What follows is a summary of major initiatives, along with example of their best practices.

17.4.1 Commercial Best Practices

- Contracting policies and practices.
- Performance and commercial specifications and standards.
- Budget policies.
- Establishing fair and reasonable prices without cost data.
- Maintenance of long-term relationships with quality suppliers.
- Acquisition of commercial and non-developmental items (including components).

17.4.2 Contracting Best Practices

- Commercial specifications and standards.
- COTS and NDI.
- Best value evaluation and award criteria.

- Open systems.
- Past performance.
- Performance-based service contracting.
- Performance-based specifications.
- Software Capability Evaluations (SCEs).
- Paperless contracting.

17.4.3 Management Best Practices

- Cost as An Independent Variable (CAIV).
- Integrated product design and development (IPDD).
- Integrated process and product design (IPPD)
- Integrated product teams (IPTs).
- Simulation Based Acquisition (SBA).
- Total Cost of Ownership.
- Earned Value Management System (EVMS).

17.4.4 Performance-Based Business Environment

- Dual-use products and processes.
- World-class processes.
- Commercial state-of-the-art technology.
- Integrate commercial and military development.
- Better, faster, cheaper, smoother.
- Integrate commercial efficiencies.

17.4.5 Defense Reform Initiative

- Focus the enterprise on a unifying vision.
- Commit the leadership team to change.
- Focus on core competencies.
- Streamline organizations for agility.
- Invest in people.
- Exploit information technology.
- Break down barriers between organizations.

17.4.6 Software Acquisition Best Practices Initiative

- Focusing the DoD acquisition community on effective, high-leverage software acquisition management practices.
- Enabling program managers to focus their software management efforts on producing quality software.

- Enabling program managers to exercise flexibility in implementing best practices within disparate corporate and program cultures.
- Providing program managers and staff with the training and tools necessary to effectively use and achieve the benefits of these practices.

17.4.7 Software Program Managers Network (SPMN) (Now under OSD-ATL Software Intensive Systems)

Note that this effort has a web site: www.spmn.com

- Productivity.
- Development and/or sustainment cost.
- Schedule.
- Quality.
- User satisfaction.
- Cost and schedule predictability.

17.4.8 Information Technology Management Reform Initiatives

- Fielding software-intensive technologies quickly, orderly, and efficiently.
- Streamlined acquisition processes.
- Use of COTS products and services, and outsourcing.
- View information systems an investments.
- Training work force in new technologies and processes.
- Protect information.
- Management best practices.

17.4.9 Single Process Initiative

- Government cost accounting standards
- The requirement to provide product cost data.
- Record keeping and reporting requirements.
- Audit and oversight requirements.
- Access to competitively sensitive financial data.
- Socioeconomic and mandatory source requirements.
- Requirements for rights in technical data.
- Security requirements.
- DoD-unique product and process specifications and standards.

As an example, under the Single Change Initiative the contract block change has been implemented to streamline the approval process for the use of commercial specifications, standards, or other processes. Figure 17-3 provides an overview of this process.

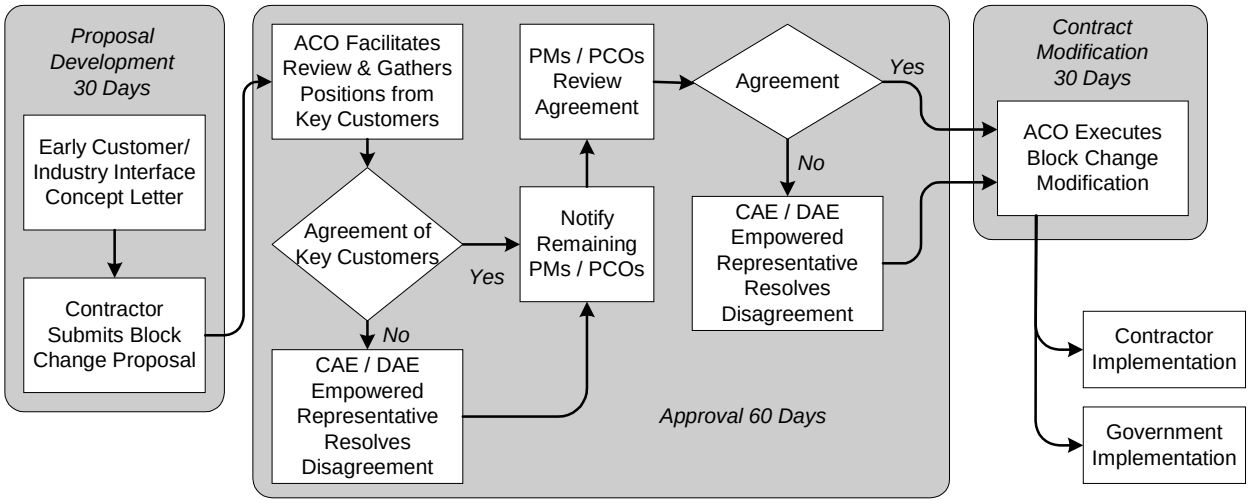

Figure 17-3 Block Change Process Overview

17.4.10 Summary

What all this means is that everything is being looked at for ways to improve the acquisition environment. Nothing is being held sacrosanct. Anything that can reduce the time or cost required to get better systems into the hands of the warfighter is a candidate for change. "We've always done it that way," is not an acceptable response to the question, "Why?" There is far too much at stake not to pursue the best acquisition environment possible. What this means to you is a responsibility to look for ways to make things better in any arena that you observe or influence.

17.5 Acquisition Environment and Regulations Checklist

This checklist is provided to assist you in better understanding your project in relation to the DoD acquisition environment. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

- ❑ 1. Do you know what mission your project or acquisition supports?
- ❑ 2. Do you understand which FAR regulations apply to your project?
- ❑ 3. Do you understand which aspects of DoD 5000.1 apply to your project?
- ❑ 4. Do you understand which aspects of DoD 5000.2-R apply to your project?
- ❑ 5. Does your project implement an architecture that supports open systems, COTS, etc? (See 17.2.3.2)
- ❑ 6. Does your project try to identify and exploit software for reuse?
- ❑ 7. Are your language choices based on sound system and software engineering principles?
- ❑ 8. Are you employing contractors who have domain experience with similar projects, a successful past performance record, demonstrable development capability, and a mature development process?
- ❑ 9. Are you striving to implement best practices in all aspects of your development/acquisition project?

17.6 References

[1] *Guidelines for the Successful Acquisition and Management of Software-Intensive Systems (GSAM)*, Version 3.0, Chapters 3 & 4, OO-ALC/TISE, May 2000. Download at: www.stsc.hill.af.mil/resources/tech_docs/

[2] Bergey, John K., et al, "The DoD Acquisition Environment and Software Product Lines", May 1999: www.sei.cmu.edu/publications/documents/99.reports/99tn004/99tn004abstract.html

17.7 Resources

Federal Acquisition Regulation and DoD FAR Supplement: http://farsite.hill.af.mil

Crosstalk Magazine: www.stsc.hill.af.mil/crosstalk/

- "Integrating Acquisition with Software and Systems Engineering": www.stsc.hill.af.mil/crosstalk/1999/08/publisher.asp
- "Outsourcing Requires More Acquisition Training": www.stsc.hill.af.mil/crosstalk/1997/09/publisher.asp
- "Evolutionary Acquisition and Spiral Development": http://www.stsc.hill.af.mil/crosstalk/2002/08/easd.html
- "Improving the Software Acquisition Process": www.stsc.hill.af.mil/crosstalk/1997/03/improving.asp

Data and Analysis Center for Software (DACS) and the Defense Software Collaborators (DSC) combined website: www.dacs.dtic.mil

DoD 5000 Series information – Latest information: http://dod5000.dau.mil

DoD Directives and Instructions: www.dtic.mil/whs/directives/index.html

Program Managers Guide to Software Acquisition, SPMN publication: www.spmn.com/products_guidebooks.html

Software Tech News magazine, subscription application: *www.dacs.dtic.mil/awareness/newsletters*

This page intentionally left blank.

Appendix

Condensed GSAM Handbook Checklists

I. Project Management Checklist (Chapter 1)

This checklist is provided to guide you in essential actions to ensure your project is on track in meeting cost, schedule, and performance requirements. If you cannot check an item off as affirmative, you need to either rectify the situation or develop a contingency plan to solve problems that may arise. For example, if the staff does not have sufficient technical skill to do the work, you will need to remedy the situation by providing training, or by obtaining sufficiently skilled people.

Beginning a Project

- ❑ The project has specific goals to accomplish and you understand the reasoning behind them.
- ❑ All stakeholders (interested parties) understand and agree on the expected project outcomes.
- ❑ Upper management is solidly behind the project.
- ❑ You understand the level of authority you have been granted in relation to the project and the rest of the organization and the level of authority is appropriate.
- ❑ You understand how the organization operates, including how to get things done within the organization.
- ❑ You understand what you are responsible for delivering at both a macro and a micro level.
- ❑ You know the high-priority risks your project faces.

During Project Planning

- ❑ You know which external interfaces are not under your control.
- ❑ You know the estimated size of the software to be developed, and how the estimate was made.
- ❑ Funding has been allocated for the project.
- ❑ A credible budget has been prepared, based on project scope and work estimates.
- ❑ Adequate time has been allocated to complete the project.
- ❑ Adequate staff is or will be available to complete project tasks.
- ❑ The project staff has sufficient expertise to perform the work.
- ❑ Facilities and tools are or will be available for the project team.
- ❑ You know of potential funding cuts and when they might come.
- ❑ You know what major problems have plagued projects of this type in the past.
- ❑ An appropriate life cycle has been selected for the project and you understand that life cycle.
- ❑ You have a credible Work Breakdown Structure (WBS).
- ❑ All requirements have work tasks assigned to fulfill them.
- ❑ All work tasks are associated with project requirements or support activities.
- ❑ Special requirements or constraints are documented.
- ❑ You have a budget, schedule, and performance baseline established and documented.

- ❑ You have identified the critical path for the project.
- ❑ You have a process established to monitor the project and detect problems and departures from the baseline.

During Project Execution

- ❑ You know what your project's expenditures are to-date and any difference between that and your budget.
- ❑ You know the status of project activity completion along the critical path and any difference between that and the schedule.
- ❑ You are aware of any issues or problems with quality or performance that may impact the critical path.
- ❑ You are aware of any contract performance issues.

II. Software Life Cycle Checklist (Chapter 2)

This checklist is provided to assist you in choosing an appropriate life cycle for your project if you are beginning a development effort, or to ensure you understand your development life cycle if your project is already under way. If you cannot check an item off as affirmative, you need to rectify the situation yourself or get help in that area.

Beginning a Development Project

- ❑ Do you have an understanding of common life cycle models, along with their strengths, weaknesses, and constraints?
- ❑ Has the operational concept been analyzed to determine what life cycle method would best support the acquisition?
- ❑ Can the requirements be fully defined prior to the beginning of the project? Are they stable?
- ❑ Do you know the timeline for deployment of the new system?
- ❑ Are funds secure for development of the entire system?
- ❑ Are the risks identified?
- ❑ Will risks impact the ability of the project to move forward at crucial points in the system development?
- ❑ Is new or developing technology to be used in the system?
- ❑ Will there be a parallel hardware development effort?
- ❑ Do you understand the level of complexity of the system to be developed?
- ❑ Do you know what the interfaces to existing and future systems are?
- ❑ Do you know the size and magnitude of the development effort?
- ❑ Do you understand the users' needs?
- ❑ Are users able or expected to participate in the development?
- ❑ Do you know what types of acquisition contracts are available for this effort?
- ❑ When choosing a life cycle model, do you know why you are choosing it over other models?

Development Project is Under Way

- ❑ Do you know what life cycle model was selected for your project?
- ❑ Do you understand the project aspects pertaining to the life cycle:
  - ❑ Phases – What are they and what are they supposed to accomplish?
  - ❑ Milestones – What are they? What is their significance?
  - ❑ Criteria for transitioning from one phase to another?
  - ❑ Deliverables – What is expected, during phases, and at the end of the project?

- ❑ Reviews – What is reviewed when? Who are the reviewers? What actions follow a successful review, an un-successful review? What are the entry and exit criteria for each review?
- ❑ Feedback mechanisms – How is feedback obtained? Who provides it? Who receives it? How is it used?
- ❑ Documentation – What is to be produced and what it is used for?

❑ Do you know where your project is in the life cycle?

❑ Is your project following the life cycle?

III. Planning Checklist (Chapter 3)

This checklist is provided as to assist you in developing a project plan. If you cannot check an item off as affirmative, you need to either rectify the situation or develop a contingency plan to solve problems that may arise.

Before Planning

❑ 1. Have you published a list of contacts for all stakeholders and team members?

❑ 2. Do you have adequate, unambiguous project objectives or requirements to work with?

❑ 3. Do you have an action plan for developing the project plan, including a schedule and a responsibility matrix for participants in the planning activities.

❑ 4. Do you know which activities depend on outputs from other activities?

❑ 5. Have you documented the planning process?

❑ 6. Have you documented all constraints and assumptions?

❑ 7. Do you have a written outline listing the contents of your project plan?

❑ 8. Have you chosen computer based tools for planning and managing the project?

During Planning

❑ 9. Are project developers included in the planning process wherever possible?

❑ 10. Are you avoiding *paralysis by analysis*, where things are studied to such a level of detail that action never takes place?

❑ 11. Are you following your planning process?

❑ 12. Are the defined tasks unambiguous?

❑ 13. Does each task have a single point of responsibility?

❑ 14. Can each task be performed by an individual or a single team?

❑ 15. Is each task associated with a single, continuous time frame?

❑ 16. Have you considered holidays, vacations, and training in your schedule and staffing plans?

❑ 17. Have you considered project management activities such as planning, meetings, and managing people?

❑ 18. Have you considered overhead time such as system down time, outages, or system repairs?

After Planning

❑ 19. Is your plan realistic, that is, achievable?

❑ 20. Are all stakeholders and participants committed to supporting the project objectives?

❑ 21. Have all involved parties formally agreed with the project plan?

❑ 22. Does your project scope or any of the objectives need to be modified?

❑ 23. Have you documented lessons learned from the planning process?

IV. Requirements Engineering Checklist (Chapter 4)

This checklist is provided to assist you in requirements engineering. If you cannot check an item off as affirmative, you need to either rectify the situation or develop a contingency plan to solve problems that may arise.

Requirements Development

- ❑ 1. Have you had extensive user involvement in developing the requirements?
- ❑ 2. Do all stakeholders understand and agree on how the system will be used?
- ❑ 3. Are all stakeholders satisfied with the requirements?
- ❑ 4. Do the developers understand the requirements?
- ❑ 5. Are all requirements clear and unambiguous?
- ❑ 6. Have you distinguished between needs and wants? Are requirements relevant?
- ❑ 7. Are requirements consistent with each other (i.e., they don't conflict.)
- ❑ 8. Are requirements complete? Do the requirements cover everything that is supposed to be accomplished?
- ❑ 9. Has design detail been left out of the requirements?
- ❑ 10. Are all requirements testable?
- ❑ 11. Can you see the requirement as an output?
- ❑ 12. Are all requirements easily recognized as requirements by using the word shall?
- ❑ 13. Have the requirements been prioritized?
- ❑ 14. Are requirements feasible with respect to cost, schedule, and technical capability?
- ❑ 15. Are requirements verifiable?
- ❑ 16. Is the system boundary clearly defined; what is in scope, what is not?
- ❑ 17. Are all external interfaces to the system clearly defined?
- ❑ 18. Is the specification written so that it can be modified when necessary, with minimal impact to the rest of the document?
- ❑ 19. Are you conducting formal and informal reviews of requirements documents?

Requirements Management

- ❑ 20. Have all requirements been entered into the requirements database?
- ❑ 21. Are the requirements traces sorted to allow requirements lookup by paragraph number, requirement number, or other useful index?
- ❑ 22. Can all requirements be traced to original system-level requirements?
- ❑ 23. Are all system-level requirements allocated to lower level, subsystem requirements?
- ❑ 24. Do you have a requirements change process documented and in place?
- ❑ 25. Have you identified members of the requirements change board?
- ❑ 26. Is adequate impact analysis performed for proposed requirements changes?
- ❑ 27. Do you know who is responsible for making the changes?
- ❑ 28. Have requirement changes been traced upward and downward through the higher and lower-level specifications?
- ❑ 29. Do you have a process in place to maintain and control the different versions of the requirements specification?

V. Risk Management Checklist (Chapter 5)

This checklist is provided as to assist you in risk management. If you answer "no" to any of these questions you should examine the situation carefully for the possibility of greater risks to the project. This is only a cursory checklist for such an important subject. Please see the reference documents for more detailed checklists.

- ❑ 1. Do you have a comprehensive, planned, and documented approach to risk management?
- ❑ 2. Are all major areas/disciplines represented on your risk management team?
- ❑ 3. Is the project manager experienced with similar projects?
- ❑ 4. Do the stakeholders support disciplined development methods that incorporate adequate planning, requirements analysis, design, and testing?
- ❑ 5. Is the project manager dedicated to this project, and not dividing his or her time among other efforts?
- ❑ 6. Are you implementing a proven development methodology?
- ❑ 7. Are requirements well defined, understandable, and stable?
- ❑ 8. Do you have an effective requirements change process in place and do you use it?
- ❑ 9. Does your project plan call for tracking/tracing requirements through all phases of the project?
- ❑ 10. Are you implementing proven technology?
- ❑ 11. Are suppliers stable, and do you have multiple sources for hardware and equipment?
- ❑ 12. Are all procurement items needed for your development effort short-lead time items (no long-lead items?)
- ❑ 13. Are all external and internal interfaces for the system well defined?
- ❑ 14. Are all project positions appropriately staffed with qualified, motivated personnel?
- ❑ 15. Are the developers trained and experienced in their respective development disciplines (i.e. systems engineering, software engineering, language, platform, tools, etc.?)
- ❑ 16. Are developers experienced or familiar with the technology and the development environment?
- ❑ 17. Are key personnel stable and likely to remain in their positions throughout the project?
- ❑ 18. Is project funding stable and secure?
- ❑ 19. Are all costs associated with the project known?
- ❑ 20. Are development tools and equipment used for the project state-of-the-art, dependable, and available in sufficient quantity, and are the developers familiar with the development tools?
- ❑ 21. Are the schedule estimates free of unknowns?
- ❑ 22. Is the schedule realistic to support an acceptable level of risk?
- ❑ 23. Is the project free of special environmental constraints or requirements?
- ❑ 24. Is your testing approach feasible and appropriate for the components and system?
- ❑ 25. Have acceptance criteria been established for all requirements and agreed to by all stakeholders?
- ❑ 26. Will there be sufficient equipment to do adequate integration and testing?
- ❑ 27. Has sufficient time been scheduled for system integration and testing?
- ❑ 28. Can software be tested without complex testing or special test equipment?
- ❑ 29. Is the system being developed by a single group in one location?
- ❑ 30. Are subcontractors reliable and proven?
- ❑ 31. Is all project work being done by groups over which you have control?

- ❑ 32. Are development and support teams all collocated at one site?
- ❑ 33. Is the project team accustomed to working on an effort of this size (neither bigger nor smaller?)

VI. Cost Management Checklist (Chapter 6)

This checklist is provided as to assist you in cost management. Consider your answers carefully to determine whether you need to examine the situation and take action.

- ❑ 1. Is cost management planning part of your project planning process?
- ❑ 2. Have you established a formal, documented cost management process?
- ❑ 3. Do you have a complete and detailed WBS, including management areas (See Mil-HDBK-881, Appendix H)?
- ❑ 4. Do you have historical information, including costs, from previous similar projects?
- ❑ 5. Have you identified all sources of costs to your project (i.e. different types of labor, materials, supplies, equipment, etc.)?
- ❑ 6. Have you identified proven and applicable estimating methods, models, and/or guides?
- ❑ 7. Have you selected computer software to assist you in estimating, budgeting, tracking, and controlling costs?
- ❑ 8. Do you have justifiable reasons for selecting your methods, models, guides, and software?
- ❑ 9. Are cost issues adequately addressed in your risk management plan?
- ❑ 10. Do you have a working change control process in place?
- ❑ 11. Does the change control process adequately address cost impact?
- ❑ 12. Do your estimates cover all tasks in the WBS?
- ❑ 13. Do you understand your project's funding profile, i.e. how much funding will be provided? At what intervals? How sure is the funding?
- ❑ 14. Have you developed a viable cost baseline that is synchronized with the project schedule and funding profile?
- ❑ 15. Do you have adequate flexibility in the cost baseline?
- ❑ 16. Do you have a plan/process for dealing with variances between cost performance and the baseline?
- ❑ 17. Have you considered incorporating earned value management into your cost management efforts?
- ❑ 18. Are you keeping records of your cost management activity for future efforts?

VII. Time and Schedule Checklist (Chapter 7)

This checklist is provided as to assist you in Time and Schedule Management. Consider your answers carefully to determine whether you need to carefully examine the situation and take action.

Preparing for Schedule Development

- ❑ 1. Have you identified an experienced, knowledgeable team to develop the schedule?
- ❑ 2. Has a process to develop the project schedule been defined and documented?
- ❑ 3. Are all time and date requirements for the project known and documented? (What is needed when?)
- ❑ 4. Are there unreasonable time constraints for the project?
- ❑ 5. Are resource capabilities and availability known?
- ❑ 6. Do you have the project constraints, assumptions, and risk plan documented?

- ❑ 7. Do all deliverables listed in the WBS have adequate and appropriate activities identified to produce them?
- ❑ 8. Have you chosen an appropriate project management software package, and are you experienced or have you been trained in using it?
- ❑ 9. Is historical duration data available for project activities?

Schedule Development

- ❑ 10. Have you identified appropriate methods and models for estimating activity duration?
- ❑ 11. Have all activities been sequenced by putting them into an activity network and indicating the dependencies between them?
- ❑ 12. Have durations been estimated for all activities?
- ❑ 13. Have the activity durations been reviewed by people experienced in those activities?
- ❑ 14. Has the critical path been identified?
- ❑ 15. Has float time been documented for all activities not on the critical path?
- ❑ 16. When developing the schedule, are you using resource leveling and remembering holidays, vacations, and sick time?
- ❑ 17. Have you developed and documented a reality-based schedule?
- ❑ 18. Have you built a time reserve into your schedule for contingencies and unforeseen events?
- ❑ 19. Has your schedule been entered into a program management software package?

Schedule Control

- ❑ 20. Have you developed and documented a schedule management plan?
- ❑ 21. Do you know how, what, when, why, and how much to monitor for schedule control?
- ❑ 22. Are you being proactive vs. reactive in your approach to schedule control by looking ahead and asking what could go wrong?
- ❑ 23. Do you have a schedule change process documented and implemented?
- ❑ 24. Are you monitoring all preparatory actions, acquisitions, deliveries, and resources for each activity to make sure they are all complete and ready when it is time to begin the activity?
- ❑ 25. Are you using experienced people to make and review schedule progress reports?
- ❑ 26. Have the various groups and individuals been given sufficient levels of responsibility, accountability, and authority to perform their tasks? Have they agreed to their assigned roles?
- ❑ 27. Are you employing regular formal and informal schedule monitoring and progress reports
- ❑ 28. Are you constantly aware of project milestones and your schedule progress?
- ❑ 29. Are you solving schedule problems by being creative and using common sense vs. extending the schedule?
- ❑ 30. Are you documenting your progress, problems, issues, solutions, and lessons learned?

VIII. Measurement and Metrics Checklist (Chapter 8)

This checklist is provided to assist you in developing a metrics program, and defining and using metrics. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action. The checklist items are divided into three areas: developing, implementing, and reviewing a metrics program.

Developing a Metrics Program

- ❑ 1. Is your use of metrics based on a documented metrics program plan?
- ❑ 2. Are you using the GQM paradigm in developing your metrics?

- ❑ 3. Are your metrics based on measurable or verifiable project goals?
- ❑ 4. Do your goals support the overall system-level goals?
- ❑ 5. Are your goals well defined and unambiguous?
- ❑ 6. Does each question elicit only information that indicates progress toward or completion of a specific goal?
- ❑ 7. Can questions be answered by providing specific information? (Is it unambiguous?)
- ❑ 8. Do the questions ask for all the information needed to determine progress or completion of the goal?
- ❑ 9. Is each metric required for specific decision-making activities?
- ❑ 10. Is each metric derived from two or more measurements (e.g. Remaining budget vs. schedule)?
- ❑ 11. Have you documented the analysis methods used to calculate the metrics?
- ❑ 12. Have you defined those measures needed to provide the metrics?
- ❑ 13. Have you defined the collection process (i.e. what, how, who, when, how often, etc.)?

Metrics Program Implementation

- ❑ 14. Does your implementation follow the metrics program plan?
- ❑ 15. Is data collected the same way each time it is collected?
- ❑ 16. Are documented analysis methods followed when calculating metrics?
- ❑ 17. Are metrics delivered in a timely manner to those who need them?
- ❑ 18. Are metrics being used in the decision making process?

Metrics Program Evaluation

- ❑ 19. Are the metrics sufficient?
- ❑ 20. Are all metrics or measures required, that is, non-superfluous?
- ❑ 21. Are measurements allowing project work to continue without interference?
- ❑ 22. Does the analysis produce accurate results?
- ❑ 23. Is the data collection interval appropriate?
- ❑ 24. Is the metrics program as simple as it can be while remaining adequate?
- ❑ 25. Has the metrics program been modified to adequately accommodate any project or organizational goal changes?

IX. Configuration Management Checklist (Chapter 9)

This checklist is provided to assist you in establishing an effective CM program. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

CM Planning

- ❑ 1. Have you planned and documented a configuration management process?
- ❑ 2. Have you identified Configuration Control Board members for each needed control board?
- ❑ 3. Has CM software been chosen to facilitate your CM process?

Establishing Baselines

- ❑ 4. Have all configuration items been identified?
- ❑ 5. Have baselines been established for all configuration items?

- ❑ 6. Has a descriptive schema been developed to accurately identify configuration items and changes to their configuration?

Controlling, Documenting, Auditing

- ❑ 7. Is there a formal process for documenting and submitting proposed changes?
- ❑ 8. Is the Configuration Control Board active and responsible in evaluating and approving changes?
- ❑ 9. Is there a "higher authority" to appeal to when the CCB gets "hung," and can't come to a consensus?
- ❑ 10. Are all changes tracked until they are fully implemented?
- ❑ 11. Are all changes fully documented in the baseline documents and change histories?
- ❑ 12. Are regular reports and configuration updates published and distributed to interested organizations?
- ❑ 13. Are regular audits and reviews performed to evaluate configuration integrity?
- ❑ 14. Are configuration errors dealt with in an efficient and timely manner?

Updating the Process

- ❑ 15. Is the CM program itself – its efficiency, responsiveness, and accuracy – evaluated regularly?
- ❑ 16. Is the CM program modified to include recommended improvements when needed?

X. Software Engineering Processes Checklist (Chapter 10)

This checklist is provided to assist you in understanding the software engineering issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

Before Starting

- ❑ 1. Do you know what software development life cycle your project will be employing and how it coordinates with the software and project life cycles?
- ❑ 2. Does the development team have experience in the software development life cycle to be used?
- ❑ 3. Do the developers, the stakeholders, and you understand what the steps of the development process are, and what the inputs and products of each step are?
- ❑ 4. Has your project been planned in the various software development areas listed in Section 10.2.1 and in Chapter 3?
- ❑ 5. Do you know what design method has been chosen for the development effort and why it was chosen over other methods?
- ❑ 6. Does your development team have experience with the chosen design method? If not, has the schedule been adjusted to allow for learning the new design method?
- ❑ 7. Have proven CASE tools been chosen to assist in the software design?
- ❑ 8. Does your development team have experience with the chosen CASE tools? If not, has the schedule been adjusted to allow for learning to use the CASE tools?
- ❑ 9. Has an appropriate programming language been chosen and do you know the reasons it was chosen?
- ❑ 10. Does your development team have experience with the chosen programming language? If not, has the schedule been adjusted to allow for learning the chosen programming language?
- ❑ 11. Is the development team sufficiently skilled and experienced in programming to properly and efficiently design, code, and test the software?

During Project Execution

- ❑ 12. Are your requirements complete, unambiguous, and agreed to by both developers and stakeholders?

- ❑ 13. Have you completed both system and functional specifications, and have they been reviewed and approved by stakeholders?
- ❑ 14. Is the development team familiar with or provided with the appropriate opportunity to become familiar with the operating system and system hardware?
- ❑ 15. Is a detailed software design being completed, reviewed, and approved before coding starts?
- ❑ 16. Is testing being properly implemented and satisfactorily completed at unit, integration, and system levels before acceptance testing?
- ❑ 17. Are human factors being considered sufficiently in the software design?

At Completion

- ❑ 18. Does the completed software correctly implement the design?
- ❑ 19. Does the software meet the requirements?
- ❑ 20. Does the software meet the users' needs?

XI. Project Health Assessment Checklist (Chapter 11)

This checklist is provided to assist you in monitoring the health of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

Before Starting

- ❑ 1. Have you prepared an overall plan for assessing project health?
- ❑ 2. Does your plan include a documented assessment process answering who, what, when, where, and how?
- ❑ 3. Does your assessment plan incorporate reviews, metrics, inspections and testing?
- ❑ 4. Do you have planned baseline budgets, schedules, etc. to compare actual project status to?
- ❑ 5. Have you scheduled regular reviews of your assessment process?
- ❑ 6. Do you have a database prepared to record project status for historical purposes?
- ❑ 7. Have you minimized the interference of your assessment activities with the project as much as possible?

Metrics

- ❑ 8. Have you developed a metrics plan as outlined in Chapter 8?
- ❑ 9. Have you selected appropriate measures and metrics for the different areas of your project (e.g. Have you implemented software development metrics applicable to your particular software efforts?)
- ❑ 10. Have you used historical data from other projects in establishing your metrics program?

Reviews [1]

- ❑ 11. Are reviews included in the project plan and schedule?
- ❑ 12. Is there a written plan for each review?
- ❑ 13. Does the plan define the scope of the review?
- ❑ 14. Does the plan specify how the review results will be documented and reported?
- ❑ 15. Does the plan identify the data to be collected?
- ❑ 16. Has the review been planned to minimize project impact?
- ❑ 17. Is a review follow up scheduled?
- ❑ 18. Does the report include recommended actions?

- ❑ 19. Are defects documented and tracked to closure?
- ❑ 20. Are all milestone stakeholders represented?
- ❑ 21. Are the Integrated Product Teams (IPTs) appropriately represented?
- ❑ 22. Does at least part of the technical review focus on software development and management?
- ❑ 23. Is the software product developed by the project suitable for its intended use?
- ❑ 24. Has risk management been addressed?
- ❑ 25. Have security issues been addressed?
- ❑ 26. Does the software comply with required standards and regulations?

Inspections

- ❑ 27. Do you have a plan for what is to be inspected, when, and by whom?
- ❑ 28. Have you used historical data to determine what is cost-effective to inspect and what is not?
- ❑ 29. Do your inspections minimize interference to the development work?
- ❑ 30. Have you emphasized inspections in the earlier stages of the project where testing cannot be performed?
- ❑ 31. Do you use the data collected from inspections to correct errors to keep them from propagating into later development stages?
- ❑ 32. Do you keep a history or database of all inspection activity, findings, and effectiveness?

Testing

- ❑ 33. Do you have a comprehensive test plan for your project, designating who, what, when, where, and how?
- ❑ 34. Have you made testing considerations a major input in the early stages of development?
- ❑ 35. Do you understand the limits of testing?
- ❑ 36. Have you implemented a testing program as outlined in Chapter 12?
- ❑ 37. Are you keeping a history of testing plans, results, and effectiveness?

XII. Testing Checklist (Chapter 12)

This checklist is provided to assist you in understanding the testing issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

Before Starting

- ❑ 1. Is testing planned for and considered throughout the entire development life cycle?
- ❑ 2. Is the overall testing strategy defined and documented, and is it an integral part of and consistent with the development program?
- ❑ 3. Is the testing process well defined, documented, understood, and supported by the development team and management?
- ❑ 4. Are test requirements clearly defined?
- ❑ 5. Are test methods, techniques, controls, and standards clearly defined and consistent with the testing strategy?
- ❑ 6. Is each test activity traceable to specific requirements?
- ❑ 7. Are configuration management and quality assurance in place and are they adequate to support the testing strategy?
- ❑ 8. Are testers trained, skilled, and motivated people?
- ❑ 9. Have adequate time and resources been reserved for testing?

- ❑ 10. Are time and resources allocated for test preparation early in the project life cycle?

During Execution

- ❑ 13. Is testing used as a primary tool to ensure good project health?
- ❑ 14. Is testing implemented as a tool for improving product quality and the development process as a whole?
- ❑ 15. Is early life cycle testing used to prevent propagation of defects to later stages of development?
- ❑ 16. Is a tracking system being used to record what has been tested and what has not?
- ❑ 17. Is a database of test results being maintained for current and future reference?
- ❑ 18. Are tests used as milestone and progress indicators?
- ❑ 19. Is the right amount of testing being done to balance risk with available time and resources?
- ❑ 20. Are you using inspections and other evaluation methods (see Chapter 11) to reduce the errors found through testing?
- ❑ 21. Do you know when your testing is complete?

XIII. Systems Engineering Checklist (Chapter 13)

This checklist is provided to assist you in understanding the systems engineering issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

Systems Engineering

- ❑ 1. Do you understand the systems engineering process?
- ❑ 2. Are you implementing an optimal systems engineering process?
- ❑ 3. Have you implemented proper and sufficient systems engineering controls and techniques?
- ❑ 4. Are you implementing systems engineering across the whole development life cycle?
- ❑ 5. Is there an experienced and skilled systems engineer directing the systems engineering effort?
- ❑ 6. Is a systems engineering representative providing input to or comments on all product change proposals?
- ❑ 7. Is the systems engineer seeing that all the various development efforts are coordinated and integrated?
- ❑ 8. Do you know what software development life cycle your project will be employing and how it coordinates with the software and project life cycles?
- ❑ 9. Are you considering all phases of the entire life cycle in your requirements, architectures, and designs?
- ❑ 10. Are you implementing an integrated product environment?
- ❑ 11. Have you established integrated (interdisciplinary) product teams?
- ❑ 12. Have you included all the necessary disciplines on the integrated product teams?
- ❑ 13. Are you documenting all studies, decisions, and configurations?
- ❑ 14. Have all internal and external interfaces been defined?
- ❑ 15. Are all your requirements verifiable?
- ❑ 16. Do all your requirements trace to products and vice versa?

COTS

- ❑ 17. Do you conduct make vs. buy vs. rent trade studies instead of just assuming that buy is the right choice?
- ❑ 18. Do you use your requirements as the criteria for your trade studies?

- ❑ 19. Do you consider the full life cycle when deciding whether to make, buy, or rent?
- ❑ 20. Do you know all your options?
- ❑ 21. Are you knowledgeable of the marketplace and the vendors?
- ❑ 22. Does the vendor understand your needs?
- ❑ 23. Are you not lowering your requirements indiscriminately to use COTS?
- ❑ 24. Do you understand the total life cycle costs?
- ❑ 25. Are the product and vendor likely to be around for the lifetime of the system?
- ❑ 26. Have you satisfactorily resolved all security issues?
- ❑ 27. Have you reduced all known risks to an acceptable level?

XIV. System Integration Checklist (Chapter 14)

This checklist is provided to assist you in understanding the system integration issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

Before Starting

- ❑ 1. Have you implemented systems engineering as an integrated life cycle effort (see Chapter 13)?
- ❑ 2. Do your test plans include and support integration efforts?
- ❑ 3. Does your development plan allocate adequate time and resources for system integration efforts, including rework time?
- ❑ 4. Are the interfaces between components, assemblies, subsystems, and systems defined in adequate detail?
- ❑ 5. Will hardware be available for testing software during integration?
- ❑ 6. Is there a contingency plan if the schedule slips if and the integration schedule is compressed?
- ❑ 7. Are all elements of the system included in the integration plan?
- ❑ 8. Is all documentation current and available for reference?

During Integration

- ❑ 9. Is there an efficient rework cycle in place to fix problems found during integration testing?
- ❑ 10. Are "fixed" modules or components integrated and retested at all levels of integration up to the level where the problem was found?
- ❑ 11. Is the people element (operators, maintainers, logisticians, trainers, etc.) being prepared to work with the system when it is deployed?
- ❑ 12. Is the support systems element (logistics, maintenance, training, etc.) being prepared to support the new system when it is deployed?
- ❑ 13. Are you following an iterative, progressive integration process?
- ❑ 14. Are experienced integrators involved with the integration?
- ❑ 15. Are area/subject matter experts involved with the integration?
- ❑ 16. Is adequate time being allowed for integration, testing, rework, reintegration, and retesting?
- ❑ 17. Are all necessary resources being made available for integration?
- ❑ 18. Is adequate testing being performed on integrated units (assemblies, subsystems, elements, system) to ensure that there are no surprises during acceptance testing?
- ❑ 19. Are you updating documentation during rework?

- ❑ 20. Are integration and system test errors being traced back to requirements and design? And if so, are the requirements and design being updated?

XV. Software Design Checklist (Chapter 15)

This checklist is provided to assist you in understanding the software design issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

Before Starting

- ❑ 1. Do you have a well-documented software development process?
- ❑ 2. Do you understand what is to be performed and produced in each phase of the design process?
- ❑ 3. Do you have a Software Standards and Conventions Document (SSCD)?
- ❑ 4. Does the SSCD contain direction in those areas listed in Section 15.2.3.1?
- ❑ 5. Are you familiar with the methods, tools, standards, and guidelines in the SSCD?
- ❑ 6. Are applicable and efficient design methods (OOD, etc.) being implemented on your project?
- ❑ 7. Are the developers experienced in the chosen development process and methods?
- ❑ 8. Is software reuse being considered throughout the development effort?
- ❑ 9. Has an analysis of alternatives been completed?
- ❑ 10. Is the selection of architecture and design methods based on system operational characteristics?

During Design

- ❑ 11. Are CASE tools being used to assist and document the design effort?
- ❑ 12. Does your design process include a robust configuration control process?
- ❑ 13. Is the design effort being properly documented? Adequate but not burdensome?
- ❑ 14. Is your team committed to following the design process?
- ❑ 15. Are all design elements traceable to specific requirements?
- ❑ 16. Are all requirements traceable to design elements?
- ❑ 17. Have all software units been identified?
- ❑ 18. Are the characteristics of all data elements identified (type, format, size, units, etc.)?

XVI. Sustainment and Product Improvement Checklist (Chapter 16)

This checklist is provided to assist you in understanding the sustainment and product improvement issues of your project. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

Sustainment

- ❑ 1. Is all your software developed with a goal to facilitate its future sustainment?
- ❑ 2. Do you understand the four types of sustainment and their purposes?
- ❑ 3. Do you understand the place and purpose of the sustainment phase in the software life cycle?
- ❑ 4. Do you understand your sustainment process?
- ❑ 5. Is there a sustainment plan?
- ❑ 6. Is there a process in place to gather problem reports and upgrade requests for the software?

- ❑ 7. Does the plan provide for reviewing, evaluating, and prioritizing upgrade requests?
- ❑ 8. Are all sustainment activity steps included in the plan?
- ❑ 9. Is there a transition plan to move to the upgraded system?
- ❑ 10. Have all activities been planned and organized to keep interference and downtime to the operating system to a minimum?
- ❑ 11. Does the plan call for running critical systems redundantly during testing and installation?
- ❑ 12. Do the deliveries include source code, documentation, and all else that is needed in addition to the software itself to continue maintaining the software?
- ❑ 13. Are all products under configuration control?

Product Improvement

- ❑ 14. Is your organization following a product improvement strategy?
- ❑ 15. Do you know where your organization is at, capability wise, relative to ISO 9001, CMMI or your chosen improvement standard?
- ❑ 16. Do you have a plan for achieving higher levels of capability?
- ❑ 17. Do your product improvement efforts emphasize improving processes to achieve product improvement?
- ❑ 18. Are your development processes documented?
- ❑ 19. Are your development processes consistent and repeated?
- ❑ 20. Does the leadership of your organization support continuous process improvement?
- ❑ 21. Is your organization committed to achieving a state of continuous improvement vs. a certificate of compliance with standards?

XVII. Acquisition Environment and Regulations Checklist (Chapter 17)

This checklist is provided to assist you in better understanding your project in relation to the DoD acquisition environment. If you cannot answer a question affirmatively, you should carefully examine the situation and take appropriate action.

- ❑ 1. Do you know what mission your project or acquisition supports?
- ❑ 2. Do you understand which FAR regulations apply to your project?
- ❑ 3. Do you understand which aspects of DoD 5000.1 apply to your project?
- ❑ 4. Do you understand which aspects of DoD 5000.2-R apply to your project?
- ❑ 5. Does your project implement an architecture that supports open systems, COTS, etc? (See 17.2.3.2)
- ❑ 6. Does your project try to identify and exploit software for reuse?
- ❑ 7. Are your language choices based on sound system and software engineering principles?
- ❑ 8. Are you employing contractors who have domain experience with similar projects, a successful past performance record, demonstrable development capability, and a mature development process?
- ❑ 9. Are you striving to implement best practices in all aspects of your development/acquisition project?