Software Extension to the PMBOK® Guide – Fifth Edition

Public Exposure Draft Review – December 2012 to January 2013

Preface

A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Fifth Edition is the recognized standard for the project management profession. The knowledge contained in the *PMBOK® Guide* has evolved from the recognized good practices of the project management practitioners who contributed to the development of the standard. In a similar manner, the knowledge contained in this extension to the *PMBOK® Guide* represents good practices of the software project management of the software project management practitioners who contributed to the development of this extension; they include the standards committee, the subject matter expert reviewers, and the public reviewers all of whom provided insightful comments. On the other hand, it must be recognized that good practices in managing software projects, as in managing other kinds of projects, must be tailored and adapted to fit the needs of each project.

It is widely acknowledged that the software medium has distinct characteristics, as compared to physical artifacts, and that those characteristics exert strong influence on the processes, methods, tools, and techniques used to manage software projects. This document, the *Software Extension to the PMBOK® Guide – Fifth Edition* thus describes the situations in which the various tools and techniques documented in the *PMBOK® Guide* are applicable for managing software projects and provides alternative approaches that are important for efficiently and effectively managing the various aspects of software projects.

This software extension is a companion document to the *PMBOK® Guide – Fifth Edition.* It is written in the style of, and follows the structure of the *PMBOK® Guide*. For consistency and ease of reference, the paragraph numbering is largely the same as in the *PMBOK® Guide – Fifth Edition*. Software project managers and other interested professionals should use the two documents concurrently.

This software extension is based on commonly accepted practices for managing software projects and on the relevant ISO/IEC and IEEE standards. These standards are cited, as appropriate, throughout this extension.

{ED NOTE: A complete list of references will be compiled prior to publication of this standard. Professionally drawn graphics will be incorporated at the time of publication.}

1 INTRODUCTION

This *Software Extension to the PMBOK® Guide – Fifth Edition* describes the generally accepted practices for managing software projects that are not common to all kinds of projects; it addresses those practices applicable for managing projects to develop new software and modify existing software. The objective of this software extension is to expand and elaborate on the general project management concepts, tools and techniques, and vocabulary found in *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Fifth Edition* **[1]**,1 and to provide more specific and precise terms, concepts, and methods for managing software projects.

Many project managers, including those certified by PMI, can improve their ability to manage projects that involve development or modification of software by increasing their understanding of the processes, methods, tools, and techniques used to manage and

successfully complete software projects as covered in this extension to the *PMBOK® Guide*. Conversely, software project managers can improve their ability to manage software projects by understanding the good practices that are documented in the *PMBOK® Guide*.

Software project managers and their project teams develop and modify software applications and the software elements of software-intensive systems. Applications programs are based on standardized hardware platforms, operating systems, and communication protocols as well as infrastructure software, such as operating system components, communication protocols, hardware drivers, and software development tools.

A software-intensive system is a collection of hardware, software, and, in some cases, operational personnel, in which software is the primary component that integrates and coordinates the operation of the system and provides a set of capabilities to a community of users. Applications programs and software-intensive systems support all aspects of modern society, ranging from information technology support for organizations, to network protocols for communications networks, to operating systems, to embedded software devices in home appliances, automobiles, mobile phones, spacecraft, consumer products, and aviation; as well as software for fields such as defense, medicine, transportation, simulation and training, banking and insurance, and recreational games.

Managers of applications and software-intensive projects face increasing challenges as software projects grow ever larger and more complex, with increasing interplay between hardware development, firmware development, software development, and the considerations related to the ergonomics of human elements within these systems.

1.1 Purpose of the Software Extension to the PMBOK® Guide

The primary purpose of the *PMBOK® Guide* is to identify and document that subset of the Project Management Body of Knowledge generally recognized as good practice. The purpose of this *Software Extension to the PMBOK® Guide – Fifth Edition* is to supplement the good practices in the *PMBOK® Guide* with knowledge and practices that can improve the efficiency and effectiveness of software project managers, their management teams, and their project members.

As stated in Section 1.1 of the *PMBOK® Guide* "good practice" does not mean the knowledge described should always be applied uniformly to all projects; the organization and/or project management team is responsible for determining what is appropriate for any given project. A similar statement applies to this software extension.

While this extension focuses on management of software projects, it will also be useful to IT organizations in many ways. First, IT organizations need to manage solution development projects. These projects may require in-house development of applications software and software-intensive systems; this extension applies directly to those projects. Second, IT organizations may outsource software development to external organizations. In these cases, this extension provides helpful information to those responsible for monitoring the external effort. The information can be used to review a third-party's project plans, analyze their progress reports and understand issues that may arise during the course of the contract. Third, most, if not all, of the organizational and team considerations explained in this document apply equally to IT technology development.

The *PMBOK® Guide* also provides and promotes a common vocabulary within the project management profession for discussing, writing, and applying project management concepts. Like all professional disciplines, software technology has a specialized vocabulary. ISO/IEC/IEEE Standard 24765 (SEVOCAB) **[2]** provides terminology for software and systems engineering. This SWX extension also provides and promotes a common vocabulary for discussing, writing, and applying software project management concepts. In cases of conflicting terminology for project management, the Glossary in the *PMBOK® Guide* and the PMI Lexicon **[3]** shall prevail. Departures from the *PMBOK® Guide* Glossary and SEVOCAB are documented in the glossary to this software extension.

The *PMBOK® Guide* also references and explains the purpose of the Project Management Institute Code of Ethics and Professional Conduct **[4]** (see www.PMI.org). For additional information on software engineering ethics, consult the IEEE Code of Ethics. In addition, the Software Engineering Code of Ethics and Professional Practice [**5**] was developed as the resource for teaching and practicing software engineering. Also, the Association of Information Technology Professionals (AITP) has developed a code of ethics [**6**]. See also the American Society for Information Science and Technology (AIS&T) Guidelines **[7]**.

1.1.1 Audience for the Software Extension to the PMBOK® Guide

The audience for this software extension includes, but is not limited to:

- 0Managers of traditional projects;
- 0Software project managers;
- 0Software team leaders;
- 0Software systems engineers;
- 0Software developers;
- 0Software V&V personnel;
- 0Software project support personnel;
- 0Software infrastructure professionals;
- 0Business analysts, business continuity planners, quality engineers, and those in related disciplines;
- 0IT CIOs, strategists, project managers, analysts, architects, solution providers, and service personnel;
- 0Program managers;
- 0Product managers;
- 0Customers;
- 0Acquirers; and
- 0Other stakeholders who affect, or are affected by, a software project.

1.2 What is a Software Project?

According to Section 1.2 of the *PMBOK® Guide*, a project is a temporary endeavor undertaken to create a unique product, service, or result. Attributes of projects, including software projects, are described in Section 1.2 of the *PMBOK® Guide*. In addition to creating new products, software projects are often undertaken to modify an existing software product, to integrate a set of existing software components, or to modify the software infrastructure of an organization. Software projects may also be undertaken to satisfy service requests, maintenance needs, and to provide operations support; however these activities may occur as low-level continuous activities; they are considered projects only when they are specified as: temporary endeavors that occur within predetermined timeframes to provide deliverables and outcomes.

1.2.1 The Relationships Among Software Portfolios, Programs, and Projects

Section 1.2.1 of the *PMBOK® Guide* describes the relationships that exist among portfolios, programs, and projects; see also Figure 1-1 of the *PMBOK® Guide*. Specifics that apply to management of software portfolios, programs, and projects are illustrated in Figure 1-1 and discussed in Section 1.4 of this software extension.

1.2.2 Why Are Software Projects Challenging?

Every discipline has unique aspects that differentiate it from other disciplines. Within those disciplines, the general principles of project management are adapted to account for the special aspects of the projects in those disciplines. Software projects are challenging for the following reasons:

- 0Software has no physical properties.
- 0Productivity of software developers is widely variable.
- 0Estimation for software projects is particularly difficult and imprecise.
- 0Risk management for software projects is predominantly process-oriented.
- 0Software is always part of a larger system. Software cannot function alone; it is executed on hardware and is often part of a larger system of diverse hardware and different users and support personnel.
- 0Software is often the most easily changed element in systems that incorporate software.

While not unique to software projects, it is certainly true that software is the product of closely coordinated intellectual teamwork, to a greater degree than occurs in many other kinds of projects. It is also true that planning of software projects is usually characterized by a high degree of uncertainty, which results from a lack of information—some people characterize software development as a learning process in which information of various kinds is gained during the project.

The primary reason why management of software projects is challenging is because software, unlike other artifacts of engineering and technology, has no physical properties that are observable by humans. Computer programs are textual representations of control signals and data flows within digital devices; the written programs are representations of and not the actual software.

Although software has no physical properties, it is a direct product of the thought processes of individuals engaged in intellect-intensive, innovative teamwork. While it is true that all engineers engage in intellect-intensive teamwork, the fact that software is developed and modified without the intervening constraints of physical media or manufacturing processes makes software teamwork different from teamwork in other engineering disciplines. As a result, many of the procedures and techniques used in software project management are designed to facilitate communication and coordination among team members engaged in closely coordinated, intellect-intensive teamwork.

Lack of physical properties has both positive and negative connotations for managing software projects. On the positive side, the intangibility of software makes it possible to rapidly respond to changing user needs and other environmental factors in comparison to changing computer hardware or other physical artifacts. On the negative side, changes need to be carefully managed; otherwise customer expectations and other stakeholder considerations overwhelm the schedule and budget constraints. “Software requirements always change” is a well-known maxim of software projects. Requirements also change for other kinds of projects but, as noted previously, software is usually easier to change than is changing the physical elements of a system undergoing development or modification; this results in frequent stakeholders’ requests for changes to software requirements. Modern life cycle processes for software projects are designed to gracefully cope with this issue.

It should be noted that stakeholder requests are not the only reason software requirements change; changes in technology, software infrastructure, business priorities, resources, and policies and regulations may necessitate changes to the software requirements.

Due to the relative ease of changing software, corrective actions that should be taken to maintain a balance among requirements (including both feature and quality and requirements), because schedule, budget, resources, and technology are sometimes lacking in software projects. This results in the perception that software projects are always behind schedule and over budget. The resulting pressure on the schedule leads to the delivery of software that has poor quality and fewer features than specified. Project management techniques for controlling software requirements are essential, as is performing impact analysis to determine and account for the effect of requirements changes on schedule, budget, resources, and technology. Changes made without careful analysis may cause unforeseeable consequences because of the interwoven aspects of software logic.

The interwoven logic of software, combined with the enormous number of computational states in even a small computer program makes exhaustive testing of software impossible for other than trivial programs. It is thus necessary for a software project manager to manage quality assurance, quality control, as well as work product verification and validation throughout the software development life cycle. As with other engineering artifacts, it is particularly important to build quality into software products rather than attempting to test it in after the fact.

Also, the lack of physical properties in software creates challenges in observing the current state of the product, which in turn complicates tracking of a software project. Traditional approaches, such as work breakdown structures, schedule networks, and earned value reporting, are tailored to fit the needs of software projects and are augmented with techniques, such as iterative development and frequent demonstrations of partially completed software.

Accurate estimation of cost and schedule is difficult for all kinds of projects, but it is particularly difficult for software projects because: (1) software is developed and modified by the intellectual work activities of the developers, (2) productivity among individual software developers varies widely (in both quantity and quality of work), (3) the requirements on which estimates are based are often poorly defined, and (4) continuing evolution of technology may make historical data inaccurate for new projects. For these reasons, modern software development life cycle methods tend to focus on developing an expanding set of product increments so that tradeoffs among schedule, budget, functionality, and quality can be continually adjusted. Controlling "scope creep" is another reason for developing and demonstrating product increments as a project evolves.

Yet another factor that creates difficulty for software projects is the degree of uncertainty in software projects. Every software project is a unique endeavor because replication (i.e., copying) of existing software is a trivial process—unlike replication of the physical artifacts of manufacturing and construction projects. Every software project is an undertaking that produces a unique product. It is said that the goal of manufacturing is to repeatedly produce artifacts that are as nearly identical as possible, given the constraints of material science and practicalities such as manufacturing technology and market acceptability; whereas the goal of a software project is to produce one perfect copy of a software product, within the constraints of schedule, budget, resources, and software technology. Management of risk and uncertainty in creating a new and different software product creates difficult challenges for software project managers.

The development of new software, or the modification of existing software may incorporate (or reuse) existing software components, but these components will likely be modified and configured in a different way to produce new or modified capabilities for users of the software. ("Users" of software to be developed or modified may include hardware, humans, and/or other software.) When existing components are reused, additional software is typically written to integrate and combine the features and quality attributes of those components; this may require significant effort because the reuse of existing software is typically not free.

Software projects are also difficult because software is always part of a larger system. Software as a stand-alone entity is useless; to be of use, software is executed on a digital hardware device. In some cases, a software project is one element of a development program that involves accompanying development of hardware components, development of related software by other projects, and development of variants of the user interface software for diverse classes of users.

In other cases, development of a software-intensive system may be limited to developing new software for a pre-specified computer, operating system, and programming language, so that the resulting system will provide capabilities for a population of users. In all cases, a software project manager is required to be aware of, and account for, the context and constraints of the larger system in which the software will be developed, deployed,

and used.

1.3 What is Software Project Management?

According to Section 1.3 of the *PMBOK® Guide*, project management is the application of knowledge, skills, tools, and techniques for project activities to meet the project requirements. Project management is accomplished through the appropriate application and integration of the 47 logically grouped project management processes comprising five Process Groups. These five Process Groups are:

- 0Initiating,
- 0Planning,
- 0Executing,
- 0Monitoring and Controlling, and
- 0Closing.

The unique nature of software, as described in Section 1.2.1, allows the elements of the 47 processes within the 5 process groups to be beneficially overlapped, interleaved, and iterated in various ways, which results in modifications of and extension to the methods, tools, and techniques in the *PMBOK® Guide* that are used to manage software projects.

According to Section 1.3 of the *PMBOK® Guide*, project management involves balancing competing constraints, which include but are not limited to:

- 0Scope,
- 0Quality,
- 0Schedule,
- 0Budget,
- 0Resources, and
- 0Risk.

Technological factors that can place constraints on software projects include:

- 0State of the art in hardware and software technology,
- 0Hardware platforms, operating systems, and communication protocols.
- 0Reuse of software components from a library,
- 0Use of open source (freely obtained) software components,
- 0Use of customer-supplied software components,
- 0Interfaces to existing software, and
- 0Hardware interfaces.

Other factors that can place constraints on software projects include but are not limited to requirements for system safety, security, reliability, availability, and information assurance, and creation of and reuse of intellectual property.

1.4 Relationship with Program and Portfolio Management

1.4.1 Software Portfolio Management

As stated in the *PMBOK® Guide,* a portfolio refers to a collection of projects or programs and other work that are grouped together to facilitate effective management of that work to meet strategic business objectives.

Organizations for which the primary mission is the development and modification of software typically treat software projects as elements of a portfolio in order to increase efficiency and effectiveness of their work activities and to engage in process improvement initiatives that will be of benefit to all projects within the organization’s portfolio. Within a portfolio, software projects are prioritized for execution based on parameters such as: complexity, degree of uncertainty, business value, returns on investment, and so

forth. Standardized life cycle frameworks that can be tailored for each project are important elements of portfolio management for software organizations. The relationship among portfolios, programs, and projects are illustrated in Figure 1-1.

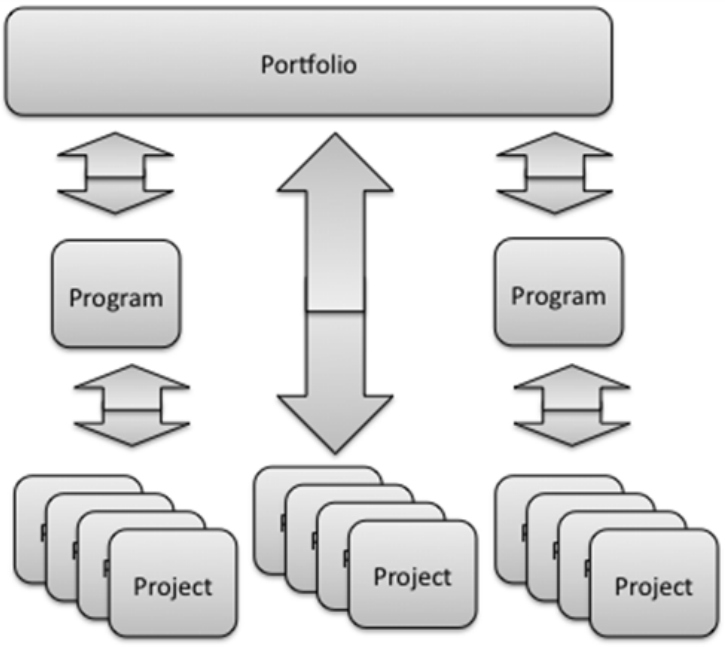


Figure 1-1. Relationships Among Portfolios, Programs, and Projects

Not all software organizations undertake programs and not all organizations manage software projects on a portfolio basis. In these cases, each software project exists as an independent entity.

Software product lines are a related concept. A product line consists of a base component that supports additions and extensions that result in specific products within the product line. For example, a product line might consist of a base component for financial accounting to which different user interfaces are added to accommodate different languages and cultures.

While software has a strong impact on organizations and their missions (both infrastructure software and applications software), there are different ways in which software can generate value, for example, financial value, social value, public welfare, and impact on work places and recreational environments. Therefore, establishing prioritization criteria for programs and projects within a portfolio may be a difficult balancing act among different value criteria.

1.4.2 Software Program Management

In some programs, software is regarded as a secondary system component; as a result, there may be no explicitly identified software project or designated software project manager. Given the central role of software in modern systems and the impact software has on system characteristics, as well as software being a primary component that can impact completion of a system on a predetermined schedule, it is essential that a designated software project manager be a member of the program management team when software is an element of a system development program.

1.4.3 Software Projects and Strategic Planning

In addition to the strategic considerations for authorizing projects, as mentioned in the *PMBOK® Guide*, software projects are sometimes undertaken to explore the feasibility of using a new development process within a specific context (such as an adaptive project life cycle model), to explore a new technology (such as cloud computing), to develop a prototype of a new style of user interface (such as a holographic or 3-dimensional display), or to exploit a software-provided innovation within itself (such as a tablet). In these cases, the business value of the software project is not the output product but the institutional learning that results from the project.

1.4.4 Software Project Management Office

In addition to the functions and objectives cited in the *PMBOK® Guide,* a project management office (PMO) for a collection of software projects may also:

- 0Provide a common repository for data relating to effort, cost, schedule, defects, and risk factors collected from software projects within an organization (i.e., within a portfolio);
- 0Use the data repository to develop one or more cost models;
- 0Use the data repository to analyze strengths and weaknesses among software projects as a basis for process improvement initiatives and for analyzing the results of process improvement activities;
- 0Assist project managers in making cost estimates and preparing project plans;
- 0Provide standard templates and forms;
- 0Acquire new software development tools and platforms for use throughout an organization;
- 0Maintain a library of reusable code modules; and
- 0Provide training for project managers and project teams.

In some organizations, the software PMO may also be involved in project management process compliance audits, project management maturity assessment, and process improvement initiatives. Project management offices, like software projects, may be subject to organizational constraints.

1.5 Relationship with Operations Mgmt and Organizational Strategy

As stated in the *PMBOK® Guide*, operations are an organizational function performing the ongoing execution of activities that produce the same product or provide a repetitive service. Operations evolve to support the day-to-day business, and are necessary to achieve strategic and tactical goals of the business. An example of operations is software production support and maintenance.

In contrast to production support in manufacturing, software production support may include supporting processes for elements such as software component integration, software configuration management, software quality assurance, software execution, and software system testing. Some or all of these supporting processes may be under the control of the software project manager, however, separate organizational units may provide a few or perhaps all of them.

1.5.1 Operational Issues and Software Project Management

Software project managers are sometimes responsible for sustaining the operation of one or more software systems while simultaneously developing a new system or a new version of an existing system. Operations personnel may report defects to be fixed in an existing system or request enhancements to an existing system. Updates in vendor-supplied software may have to be installed. Fixing defects, providing enhancements, and installing updates may divert resources from the project at hand and thus disrupt schedules and budgets.

1.5.1.1 Operations Management

As stated in the *PMBOK® Guide*, operations management is a subject area that is outside the scope of formal project management as described in the *PMBOK® Guide*.

1.5.1.2 Operational Stakeholders in Software Project Management

The *PMBOK® Guide* distinguishes between operational personnel and product users. While operations management is different from project management (see Section 1.5.1.1 of the *PMBOK® Guide*), the needs of the stakeholders who perform and conduct business operations are important considerations in managing software projects. The software supported by and used by operational personnel exerts a strong influence on the efficiency and effectiveness of their work processes and procedures; therefore the inputs of operational stakeholders are important sources of requirements that software projects strive to satisfy. It is important to address requirements that will improve the efficiency of sustainment; software projects should also consider issues related to deployment, sustainment, replacement, retirement, and disposal of the software product.

1.5.2 Organizational Issues and Software Project Management

As stated in the *PMBOK® Guide,* governance usually sets high-level strategic direction and performance parameters. The strategy provides the purpose, expectations, goals, and actions necessary to guide business pursuit, and is aligned with business objectives.

ISO/IEC/IEEE Standards 15528 [**8**] and 12207 [**9**], as well as the Capability Maturity Model Integrated for Development (CMMI-DEV) [**10**] and the Capability Maturity Model Integrated for Services (CMMI-SVC) **[11]** are used by many software organizations as frameworks for specifying strategic directions and performance parameters. They also serve as models for process and product improvement initiatives.

1.5.2.1 Software Project-Based Organizations

Many software organizations are project-based. As stated in the *PMBOK® Guide*, project-based organizations weaken the hierarchy and bureaucracy inside the organizations as the success of the work is measured by the final result rather than position or politics. A project-based organization is particularly desirable for software projects, which are often comprised of self-enabling teams of creative individuals.

1.5.2.2 Link Between Software Project Management and Organizational Governance

As stated in the *PMBOK® Guide,* projects (and programs) are undertaken to achieve strategic business outcomes, for which many organizations now adopt formal organizational governance processes and procedures.

In addition to internally imposed governance policies, many software projects are conducted for external customers who may require that certain processes and procedures be followed when developing software that may place at risk the health, safety, or welfare of the public. There may also be compliance requirements, such as Sarbanes-Oxley in the United States or the policies governing development of medical devices that contain software. The type of contract with an external customer may also affect the way in which a software project is governed.

1.5.2.3 The Relationship Between Software Project Management and Organizational Strategy

The statements in this section of the *PMBOK® Guide* apply equally to software projects, which should support the strategies and goals of the organization. Software organizations sometimes undertake contracted projects that are not aligned with the mission and goals of the organization in order to provide jobs and income. Although these projects can provide job security and, in some cases, allow survival of the organization, they often unnecessarily divert organizational assets in cases where survival is not at stake. On the positive side, a software project may be undertaken to support an organizational strategy of, for example, developing an inventory application to accommodate a new product line that supports the business goal of growing the revenue stream for that product line.

1.6 Software Business Value

According to this section of *PMBOK® Guide,* business value is defined as the entire value of the business; the total sum of all tangible and intangible elements. A software product may be a proprietary product of the business and provide a major revenue stream for that business or business unit. In some cases, infrastructure and customer support software may be capitalized and depreciated over time.

Because software is the product of closely coordinated teamwork that is often innovative, the intellectual capital of a software organization (the software developers, maintainers, and other software personnel) is a particularly important element of business value. Another important element of business value for software organizations is the set of processes and procedures that enable software project management and product development in an efficient and effective manner by enhancing communication and coordination among individuals, teams, projects, programs, customers, and other external stakeholders.

1.7 The Role of a Software Project Manager

As stated in the *PMBOK® Guide*, the role of a project manager is distinct from that of a functional manager or an operations manager. However, software project work may be internally organized functionally by product component (i.e., user interface, database, computation, and communication software) or functionally by process component (i.e., analysis, design, construction, testing, and installation/training processes). The project team may be organized in a functional manner and the software project manager may, as a result, be the manager of the project's functional units during planning and execution of that project.

In addition to the characteristics of project managers listed in the *PMBOK® Guide* (knowledge, performance, and personal), software project managers provide leadership in:

- 0Planning and estimating: both initially and on an ongoing basis as conditions change;
- 0Monitoring and controlling of schedule milestones, budget expenditures, requirements stability, staff performance, resource utilization, and identified risk factors, using a systematic version control process;
- 0Leading and directing: by defining the project vision and maintaining it as requirements and other constraints change and by providing hands on, day-to-day leadership of team leaders, software developers, and supporting personnel who are engaged in

innovative teamwork;

- 0Managing risk by identifying, analyzing, prioritizing, and responding to risk factors on an ongoing, continuous manner; and
- 0Coaching, monitoring, inspiring, and working with the software engineering knowledge workers to obtain desired results. **[19]**

On a small project (i.e., less than 10 people), the software project manager may also be the team leader, the software designer, and assume other contributing roles.

1.7.1 Interpersonal Skills of a Software Project Manager

Because of the intangible nature of software and the intellectual effort required to produce software, an important factor for project success is the communication and coordination mechanisms instituted and reinforced by the project manager, as conveyed by the interpersonal skills of the project manager. The project manager must ensure that effective communication and coordination occurs within the project team and with entities external to the project.

Interpersonal skills that are particularly important for software project managers include:

- 0Leadership,
- 0Team building,
- 0Motivation,
- 0Communication,
- 0Influencing,
- 0Decision making,
- 0Political and cultural awareness, and
- 0Negotiation.

1.8 This Software Extension

As stated in Section 1.8 of the *PMBOK® Guide,* project management standards do not address all details of every topic. This standard is limited to single projects and the project management processes that are generally recognized as good practice. Other standards may be consulted for additional information on the broader context in which projects are accomplished."

This *Software Extension to the PMBOK® Guide – Fifth Edition* is similarly limited to management of single software projects and the software project management processes that are generally recognized as good practice. Good practices outlined in this extension are generally applicable to most software projects, most of the time. In addition, this extension provides information about IEEE Computer Society standards for software and systems development that reflects standard practice in the software industry. These standards are cited throughout this software extension.

The following observations summarize some of the key points made previously and provide pointers to the following sections of this software extension.

- 0Because software is not subject to physical constraints, there are many possibilities for software project life cycles and many different approaches to applying the techniques of software project management to those life cycles. Section 2.4.2 of this software extension describes the continuum of software project life cycles that vary from predictive to iterative to highly adaptive. "Most software projects, most of the time," uses some variation within the continuum of iterative cycles for developing software between iterative-incremental and highly adaptive; software project managers adapt the methods and techniques of project management to those project life cycles.
- 0Software project managers should have a basic understanding of software engineering development processes and various approaches to managing software projects

within the continuum of software project life cycles.

- 0Software project managers may have technical backgrounds, but not always in software. Those with strong technical backgrounds may need to focus on developing their business, project management and interpersonal skills. Conversely, some software project managers may not have strong software skills and may need to work closely with a technical leader on each of their software projects.
- 0Two especially important aspects for software project managers are interpersonal skills and an understanding of software quality attributes. Section 1.7.1 lists interpersonal skills that are particularly important for software project managers.

Management of software quality is particularly important because the safety, security, and welfare of the general public are increasingly dependent on software. The deliverable work products of software projects may include artifacts such as requirements; design description; source code; object code; test results; test suites; configuration library;

the complete tool chain, including the version control tool used during software development, installation, and delivery; maintenance instructions; and operator and user instructions and training manuals and materials. Each of the deliverable work products should have acceptable quality, as determined by the needs of the users and other stakeholders. As mentioned previously, management of quality assurance, quality control, verification, and validations are especially important aspects of managing software projects.

Software quality attributes that are important to users and other stakeholders include, but are not limited to attributes such as:

- 0Safety,
- 0Security,
- 0Reliability,
- 0Performance,
- 0Ease of learning,
- 0Ease of using,
- 0Interpretation of error messages
- 0Availability,
- 0Accessibility,
- 0Efficiency,
- 0Flexibility,
- 0Interoperability, and
- 0Robustness.

Software quality attributes that are important to software developers include, but are not limited to:

- 0Testability,
- 0Maintainability,
- 0Portability,
- 0Extensibility, and
- 0Reusability.

While these quality attributes are not unique to software, it is important that a software project manager understand the priorities among necessary and desired quality attributes of the software work products and the influence that quality attributes exert on the methods, tools, and techniques used to manage and conduct software projects. These topics are addressed in Section 8 of this software extension.

2 Software Project Life Cycle and Organization

This section of the *Software Extension to the PMBOK® Guide – Fifth Edition* describes software project life cycles; how software projects interact with ongoing operational work

and other elements of the organization; the influence of stakeholders beyond the software development team; and how organizational structure affects the way a project is initiated, planned, executed, monitored and controlled, and closed. Emphasis is placed on tools and techniques that have been developed to accommodate the special and unique aspects of software projects and management of software projects.

2.1 Organizational Influences on Software Project Management

As stated in Section 2.1 of the *PMBOK® Guide*, the organizational culture, style, and structure influence how projects are performed. Organizational culture and style have a strong influence on how software projects are performed because software engineers are knowledge workers who develop software by engaging in closely coordinated teamwork.

2.1.1 Organizational Cultures and Styles for Software Projects

There are two extremes of organizational culture and style for software projects, with many variations in between. Organizations that engage in large, complex software projects tend to have more rigorously controlled processes and procedures than do organizations engaged in smaller, simpler efforts. In the former case, a software organization may have separate functional units to accomplish requirements analysis and architectural design, software construction, and verification and validation; with iterations within and among these activities, as necessary. Techniques such as work breakdown structures and earned value reporting are typically utilized on large software projects.

At the other extreme, the organizational culture and style may minimize formality in documenting requirements and design and focus on iterative development cycles in which the requirements and design emerge through frequent demonstrations of increasing software capabilities for continually engaged customers and users who provide feedback and direction.

Many variations within this continuum of software project life cycles are found within software development organizations. Also, there are other dimensions that may influence the organizational culture and style. For example, an organization that develops software for medical devices might not place undue emphasis on project planning but, because of government regulations, may have rigorous documentation requirements.

In addition, because of the unique nature of software projects (intangible product and intellect-intensive teamwork), the organizational factors that influence the morale and motivation of software workers are somewhat different than for others who work in organizations.

Organizational factors that tend to increase motivation, engagement, and productivity of software personnel include issues such as:

- 0Workplace free of disruptions and external interruptions
- 0Challenging technical problems,
- 0Autonomy to solve problems,
- 0Ability to control one’s work schedule,
- 0Learning new things,
- 0Competent technical leaders,
- 0Compelling vision or end-state, and
- 0Adequate software tools and computing technology.

Software development—whether in adaptive projects or predictive projects—tends to be a learning experience. Organizational factors that tend to increase learning among project team members and therefore increase product quality and project performance include:

- 0Collaborative culture,
- 0Easy access to cross-functional team members,

- 0Opportunities to discuss issues in a timely fashion,
- 0Access to needed information,
- 0Colocation or electronic connectivity that results in easy communication among the team members, and
- 0High level of trust among the team members and between the project team and the project manager, other managers, and the customer (whether internal or external to the organization) allowing for open discussion of challenges and options to increase the probability of project success.

Conversely, the absence of these factors can result in decreased motivation and morale, at both the individual and team levels. These factors are especially important for software workers.

2.1.2 Organizational Communications

Software organizations, like other modern organizations, utilize the communication mechanism noted in Section 2.1.2 of the *PMBOK® Guide.* In particular, the nature of software (no physical properties) and the growth of the Internet and Web infrastructure, as noted in the *PMBOK® Guide,* have made possible the globalization of software development. Software project managers increasingly manage virtual projects.

2.1.3 Organizational Structures for Software Projects

From the organizational point of view, organizations that conduct software projects may organize projects as individual entities, project-by-project (projectized organization); by coordination among functional units (functional organization); or as a matrix organization that combines projectized and functional structures.

Internally, the structure of a software project is typically organized into one or more small teams (i.e., ten or fewer members per team) where the number of teams depends on the scope of the project. Small, coordinated teams are used to minimize the problems of communication within teams because the number of communication paths within a team increases exponentially with the number of team members. Several small teams have fewer overall communication paths than one large team (see Section 5). Other functional elements of a software organization may provide supporting services such as configuration management, infrastructure tools and support, and separate verification and validation units.

When organizing a software project, it is important to align the project structure with the desired structure of the software product. As observed by Conway (in a statement that became known as Conway's Law):

"Any organization that designs a system (defined more broadly here than just information systems) will inevitably produce a design whose structure is a copy of the organization's communication structure." **[12]**

A project that is organized using three software development teams (or a single team of 3 members) will develop a software product having three components; if a project of four teams (or a single team of 4 team members) develops the same product it will likely have four components, because software is a product of the closely coordinated intellectual effort of teams and team members who divide among them the features and interfaces to be implemented.

Section 2.1.3 of the *PMBOK® Guide* provides additional information about organizational structures applicable to software projects.

2.1.4 Organizational Project Assets

According to Section 2.1.4 of the *PMBOK® Guide*, organizational process assets include any

and all process-related assets that can be used to influence the project's success, from any or all of the organizations involved in the project. In the *PMBOK® Guide* organizational project assets are grouped as processes and procedures and the corporate knowledge base. The *PMBOK® Guide* provides examples of project assets; they are applicable to software projects. Additional considerations for software projects include:

2.1.4.1 Software Processes and Procedures

The processes and procedures of many software development and service organizations are based on ISO/IEC and/or IEEE standards for software engineering and on the Capability Maturity Models and the process asset library of the Software Engineering Institute2.

Some ISO/IEC and IEEE standards have been harmonized and issued as joint standards (ISO/IEC/IEEE); they include:

- 0ISO/IEC/IEEE 15288: Systems and software engineering—System life cycle processes
- 0ISO/IEC/IEEE 12207: Systems and software engineering—Software project life cycle processes
- 0ISO/IEC/IEEE 16326: Systems and software engineering—Life cycle processes—Project management [13]
- 0The Capability Maturity Models Integrated (CMMI), developed and maintained by the Software Engineering Institute include:
- 0CMMI for Development (CMMI-DEV) **[10]**,
- 0CMMI for Services (CMMI-SVC) **[11]**, and
- 0CMMI for Acquisition (CMMI-ACQ) [**14**].

CMMI for Development is a collection of best practices for system and software engineering. CMMI-DEV, V1.3 contains 22 process areas. The process areas are grouped into four categories; one of the categories is project management, which has 7 process areas as follows:

- 0Project Planning,
- 0Project Monitoring and Control,
- 0Supplier Agreement Management,
- 0Integrated Project Management
- 0Requirements Management
- 0Risk Management, and
- 0Quantitative Project Management.

2.1.4.2 Corporate Knowledge Base for Software Development

Section 2.1.4.2 of the *PMBOK® Guide* provides examples of information that is typically contained in a corporate knowledge base. Many software organizations maintain corporate knowledge bases that contain information similar to that in Section 2.1.4.2 of the *PMBOK® Guide*.

CMMI-DEV includes the process area, Organizational Process Definition, the purpose of which is to establish and maintain a usable set of organizational process assets and work environment standards. Also, generic practice GP3.2 in CMMI-DEV (Collect Improvement Information) states information and artifacts are stored in the organization's measurement repository and the organization's process asset library."

Although many software organizations maintain a corporate knowledge base for software engineering and software project management, it is not a universal practice.

2.1.5 Enterprise Environmental Factors

The factors cited in Section of 2.1.5 of the *PMBOK® Guide* apply equally to software projects. They include, but are not limited to:

- 0Organizational culture, structure, and processes;
- 0Government or industry standards;
- 0Infrastructure (e.g., facilities and capital equipment);
- 0Human resources;
- 0Personnel administration;
- 0Political climate; and
- 0Project management information systems.

2.2 Software Project Stakeholders and Governance

2.2.1 Software Project Stakeholders

A software project stakeholder is any individual or organizational entity that affects or is affected by a software project or the resulting software product. Stakeholders include both internal and external stakeholders. Internal stakeholders include the project team and other organizational entities such as a marketing or contract administration department. External stakeholders include acquirers, integrators, customers, and users and may include policy makers and regulatory agencies.

Because of software's abstract nature, software project deliverables are subject to broader and more variable interpretations by project stakeholders than are physical entities. It is important to engage the appropriate stakeholders in issues of relevance to them as often as is appropriate in order to gauge expectations on the project deliverables and project governance. This topic is explored in depth in Section 13 of this software extension (Stakeholder Management). Stakeholder management is a project deliverable.

2.2.2 Software Project Governance

According to the *PMBOK® Guide*, project governance is the alignment of project objectives with the strategy of the larger organization by the project sponsor and project team." Governance is concerned with issues such as decision making, prioritization, and alignment of vision and strategy with an organization's work. Organizational governance for software projects may include elements such as a software project management office (SPMO), a software project portfolio management, or an IT strategy group. The intangible nature of software may result in a high level of formality in the governance model, in an attempt to bring visibility to an inherently invisible product. Software projects typically involve discovery of requirements and constraints within a learning environment as the projects evolve. Formal governance models that treat software development as a linear, predictive process may exert a detrimental impact on the software projects conducted by the organization. While different types of projects call for different levels of rigor and formality of governance, it is important that the governance models are suited to the nonlinear, adaptive learning environment of software development.

2.3 The Project Team

2.3.1 Composition of Software Project Teams

The composition of a software project team is often a balance between ideal considerations and practical constraints. Ideal considerations for software development teams include:

- 0**Dedicated vs. non-dedicated staff members.** In the world of knowledge work, switching context between multiple assignments incurs intellectual overhead. Therefore, software projects benefit from dedicated resources. Assigning team members to one project at a time can limit switching of contexts among multiple, part-time tasks and improve the productivity of software teams. However, some projects simply do not have enough work for

the various specialized skill sets required nor the budget to support dedicated resources for those specialized skills. As a result, many software project managers strike a balance between dedicated and non-dedicated resources.

- 0**Collaborative team vs. functional division**. In some organizations, dedicated team members work together with all the skills required to deliver tested working software represented within the team rather than allocating functionality to be developed among separate functional units. The latter approach may involve assigning development of user interface components to the user interface group, database components to the database group, etc. In contrast, a collaborative team may include expertise in user interfaces, databases, and other needed specialties. Aligning teams in a collaborative manner increases feedback among team members and reduces feedback time. It also allows learning to occur throughout the course of the project, which is reflected in the work products and interactions among the team members. Some software organizations maintain functional groups in the interest of maximizing utilization of specialty resources. As indicated previously, striking a balance between dedicated and non-dedicated resources, which may be reflected as a collaborative team versus functional divisions is a challenging economic dilemma in software development.

- 0**Virtual vs. colocated.** The complex and abstract nature of software makes it difficult for software engineers to communicate detailed technical issues in written form. In order to convey abstract ideas and achieve the collaboration needed for innovation, many teams benefit from the high fidelity of face-to-face conversations using white boards. An additional benefit is the tacit knowledge that is gained and used when project team discussions are held in common meeting areas. However, some organizations need to control costs by outsourcing project staff using low-cost providers and limiting travel. As a result, a project manager may choose to strike a balance between face-to-face communication for activities such as project orientation, training, and key meetings (e.g., kickoff and coordination meetings), with day-to-day work performed in a virtual environment.

- 0**Specialists vs. generalists**. Software projects often require specialized skills that incur high labor costs. As a result, many project managers staff their software projects with project members who have generalist skills and rely on them to perform the majority of the work. Then, periodically an expert will be called upon to mentor the generalists in specialized areas. Another benefit of this approach is that generalists may offer broader perspectives than experts and develop more solution options.

- 0**Stable vs. interim**. Many organizations create a new project team for each software project and disband the team when the product is delivered. For ongoing maintenance, enhancement, and support of software products, it is beneficial to keep cross-functional teams together over time so that knowledge is retained about the product, team interactions and learning are maintained and improved, and teams maintain high levels of performance. Another benefit of stable teams is that project performance throughout the organization typically becomes more predictable.

In practice, organizations may have to make trade-offs among these considerations. A detailed exploration of software project teams is included in Section 9 on Human Resources Management.

2.4 The Software Project Life Cycle – Overview

According to Section 2.4 of the *PMBOK® Guide,* project life cycles can be described as falling somewhere in a continuum from predictive or plan-driven approaches at one end to adaptive or change-driven approaches at the other.

Software has no physical properties and lends itself to a wide variety of software project models. These models occupy positions within the predictive to adaptive continuum that span from a linear sequence of project phases in which an individual phase may last for a

few weeks to a few months (highly predictive), to project models that have daily iterative cycles and include frequent demonstrations of partially completed, working software (highly adaptive). The continuum of software project life cycles is illustrated in Figure 2-1.

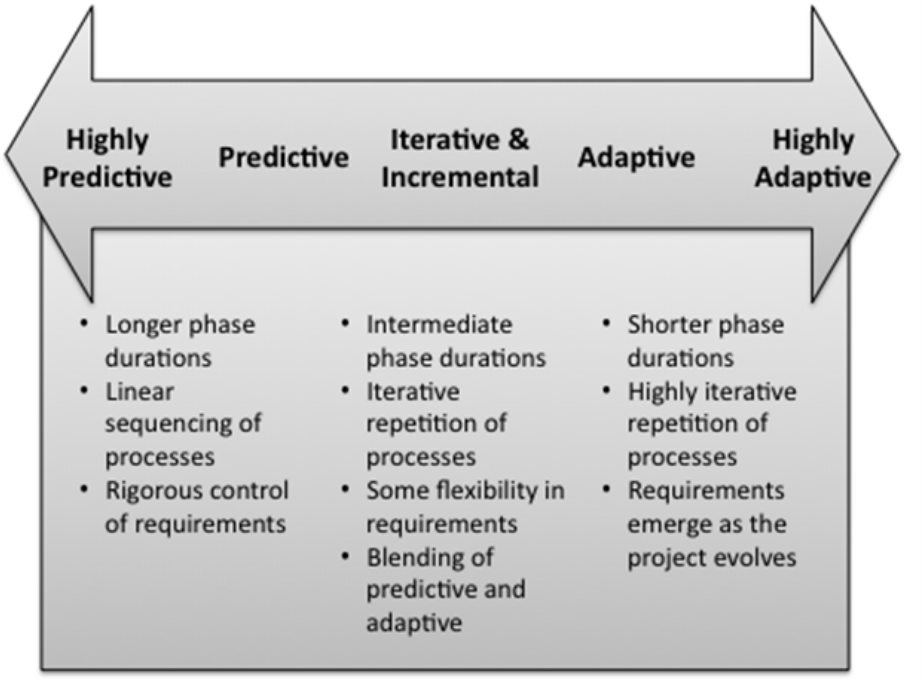


Figure 2-1. The Continuum of Software Project Life Cycles

It must be emphasized that the continuum in Figure 2-1 is not a thin straight line. Software project life cycles are complex and multidimensional and include processes, such as configuration management, process and product quality assurance, independent testing, and other processes as appropriate and needed. In addition, software projects may include interfaces to programs of which they are an element, to other parts of the software development organization, affiliated projects, or to subcontracted groups or projects.

There is a subtle distinction between a *software project life cycle* (as the term *life cycle* is used in the *PMBOK® Guide),* and the *life cycle model* that is used for a particular software project. The malleability of software permits intermixing of elements from the life cycle continuum. For example, a software project life cycle model may rigorously control requirements (predictive) but use frequent iteration cycles during software construction (adaptive). Each software project has a tailored software project life cycle model based on elements of the project life cycle continuum needed to conduct the project efficiently and effectively.

Another distinction is between *software project life cycle model* and *software product life cycle model*. A product life cycle model includes an initial project life cycle model but also includes the processes for deployment, support, maintenance, evolution, replacement, and retirement of a software product. Additionally, enhancement and adaption of the initially delivered software may involve several project life cycles beyond the initial one.

A highly predictive project life cycle model, as partially illustrated in Figure 2-2 (also known as a waterfall life cycle) is characterized by a linear sequencing of development phases and longer phase durations, as compared to an adaptive life cycle. Predictive life cycles may be appropriate for software projects that have well-defined requirements, a familiar problem domain, stable technology, a known customer (either internal or external), and a short duration (i.e., 3 months or less). However, most software projects require adaptive approaches to accommodate changing requirements, a new problem domain, new or updated technology, a new customer, or an extended duration plus the learning that

typically takes place during the project, In those cases, an adaptive life cycle model is appropriate.

As noted in the *PMBOK® Guide*, iterative and incremental life cycles are those in which the project scope is generally determined early in the project life cycle. A project life cycle can be either iterative or incremental or iterative-incremental (as illustrated in Figure 2-1). An incremental model might develop software using a sequence of development phases where each phase produces working software and where each phase might be adaptive or predictive. Iterative life cycle models for software projects typically produce working software but it is not unusual for the requirements and design phases of a predictive model to proceed iteratively before moving to the next phase of software development. See Section 2.4.2.3 for a discussion of iterative and incremental software project life cycle models.

As noted in the *PMBOK® Guide*, most software project life cycles develop the product both iteratively and incrementally. In Figure 2-1 the iterative-incremental life cycle (It-In) occupies the middle range of the life cycle continuum. An It-In life cycle model for a software project may be more on the predictive side or more on the adaptive side. For example, a life cycle model for a software project could be based on the It-In life cycle and instantiated as a project life cycle model having three iterative cycles with the requirements to be revised at the end of the first two iterations based on demonstrations of working software. In this case, the project life cycle model would lie on the adaptive side of the continuum. On the other hand, adherence to rigorously specified and rigorously controlled requirements would place the It-In project life cycle model on the predictive side of the continuum.

Adaptive life cycles for software projects are depicted in Figure 2-1. According to the *PMBOK® Guide* adaptive life cycles (also stated as change-driven or agile methods in *PMBOK*) are intended to facilitate change and require a high degree of ongoing stakeholder involvement. There are several adaptive software project life cycle models that incorporate various elements of agility. See Section 2.4.2.4 for a discussion of adaptive life cycle models for software projects.

Highly adaptive software project life cycles are exploratory and experimental in nature. In exploratory development, a hypothesis concerning a possible solution is postulated and software is constructed to test the hypothesis. The solution is tested in use to gain rapid feedback on the hypothesis. Based on the feedback, the solution may be discarded, further developed in a highly adaptive fashion, or further developed using a different software project life cycle model. A sequence of increments may be constructed while testing the hypothesis; development of increments may all be based on the same life cycle model throughout or different increments may be developed using different software project life cycle models.

Software prototyping is a related concept and often used to present alternative user interface designs to user representatives or to explore a technical option. Software prototypes differ from prototypes of physical products, which are often fully functional units. In contrast, a software prototype is typically a rapidly developed mockup of one or a few software features. Software prototyping is a valuable technique that can be used at any time during software development but software prototypes are not constructed to be deliverable software products and software prototyping is not a software project life cycle model suitable for developing a software product. Software project managers should take care to ensure that prototype software that has not been productized (i.e., well structured, documented, and thoroughly tested) is not incorporated into a software product. Software prototypes are often designated as “throwaway prototypes” to maintain this distinction.

2.4.1 Characteristics of the Software Project Life Cycle

The creation of software deliverables typically requires a variety of project life cycle processes. According to ISO/IEC/IEEE Standard 12207 development of software includes the following processes:

- 0**Analyze**: Software Requirements Analysis Process (refer to Section 7.1.2 of ISO/IEC/IEEE Standard 12207)
- 0**Architect**: Software Architectural Design Process (refer to Section 7.1.3 of ISO/IEC/IEEE Standard 12207)
- 0**Design**: Software Detailed Design Process (refer to Section 7.1.4 of ISO/IEC/IEEE Standard 12207)
- 0**Construct**: Software Construction Process (refer to Section 7.1.5 of ISO/IEC/IEEE Standard 12207)
- 0**Integrate**: Software Integration Process (refer to Section 7.1.6 of ISO/IEC/IEEE Standard 12207)
- 0**Test**: Software Qualification Testing (refer to Section 7.1.7 of ISO/IEC/IEEE Standard 12207)

For purposes of exposition, this extension will refer to these software implementation processes to describe the variations in software project life cycles found in the software industry.

Figure 2-9 in the *PMBOK® Guide,* illustrates typical cost and staffing levels across the project life cycle. The build-up and build-down of cost and staffing level for the "carrying out the work" and "closing" segments of Figure 2-9 are typical for predictive software project life cycles but these segments of the figure are typically flat for adaptive software project life cycles.

2.4.2 Software Project Phases

2.4.2.1 Phase-to-Phase Relationships

According to the *PMBOK® Guide,* there are two basic types of phase-to-phase relationships: sequential and overlapped. The volatile nature of software allows for significant flexibility in overlapping, interleaving, and iterating software development phases. ISO/IEC/IEEE Standards 15288 and 12207 use the term *stages* rather than *phases* to indicate that these stages are to be used throughout a project whenever they are needed to achieve project objectives. Highly adaptive software project life cycle models, for example, execute many of the processes during each iterative cycle, as explained in Section 2.4.2.4.

2.4.2.2 Predictive Software Project Life Cycles

The *PMBOK® Guide* defines predictive life cycles as those for which the project scope, and the time and cost required to deliver that scope, are determined as early as practically possible in the project life cycle.

When using a predictive life cycle, each project phase is organized around a specific activity within the project life cycle. Namely, the first phase is dedicated to defining the product concept and collecting requirements, which in turn leads to a dedicated phase of defining the requirements to be included in the product. The requirements phase is followed by a dedicated phase to specify the design of the entire software deliverable. The subsequent phases (construction, integration, testing, etc.) follow in sequence. A highly predictive life cycle does not allow any work activities in subsequent phases until the current phase is completed.

Variations on a highly predictive project life cycle allow some work to proceed on one or more well defined subsystems while earlier work continues on other subsystems, or to

continue into the next phase while some documented problems are corrected. Figure 2-2 illustrates the sequential phases of the highly predictive life cycle for software projects; the feedback that may occur between the phases is not illustrated.

The graphic above each project phase is similar to Figure 2-11 of the *PMBOK® Guide*, which illustrates the application of the five Process Groups within each project phase (Initiating, Planning, Executing, Monitoring and Controlling, and Closing).

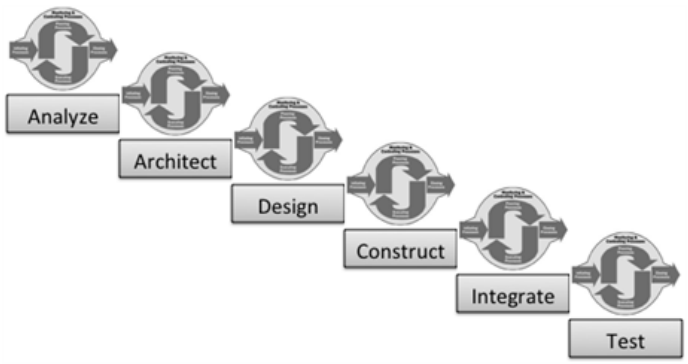


Figure 2-2. Sequential Phases of a Predictive Software Project Life Cycle

As stated in Section 2.4 of this extension, a predictive software project life cycle may be appropriate for software projects that have well-defined requirements, a familiar problem domain, stable technology, a known customer (either internal or external), and a short duration (i.e., 3 months or less). Also, a predictive software project life cycle model may be used when specialized resources required for a project phase are only available for a limited time. It may also be appropriate when trying to achieve short-term efficiencies in specific software activities. Additionally, this approach may be used on routine projects, where the requirements, technology, and customer are familiar and when there is less need to gain new understanding or accommodate ongoing changes.

However, due to the typically volatile nature of software requirements, many software development organizations and software projects use iterative, incremental, and adaptive approaches. When working with complex software projects, the need to change some element of a previously completed activity phase occurs frequently because: (1) the requirements are emergent in nature, (2) new understandings arise around stakeholder expectations regarding scope, (3) new insights into the technology are discovered, or (4) errors in previous work need to be fixed. As a result, the intended isolation of a specific activity to a specific project phase becomes increasingly impractical as product size and complexity increase.

2.4.2.3 Iterative and Incremental Software Project Life Cycles

According to Section 2.4.2.3 of the *PMBOK® Guide,* iterative and incremental life cycles are ones in which the project scope is generally determined early in the project life cycle, but time and cost estimates are routinely modified as the project team understanding of the product increases. Iterations develop the product through a series of repeated cycles, while increments successively add to the functionality of the product.

Most life cycles develop the product both iteratively and incrementally.

For software projects, it is not uncommon that some or all of the time, cost, or requirements are modified as the project team's understanding of the software product increases, thus providing a three-way trade-off space. In some cases one or more of the three may be fixed, which restricts the tradeoff space. Typically, attempting to fix all three (time, cost, and requirements) without any flexibility results in software project failure. The information in Section 2.4.2.3 of the *PMBOK® Guide* concerning iterative and incremental life cycles is generally applicable to software projects.

- 0***Iterative project life cycle phases.*** Iterative software project life cycle phases systematically repeat one or more of the software development stages, thus iteratively converging on a product that satisfies the defined scope. The software product is progressively elaborated, and incorporates feedback as new information or understanding emerges during the project. This approach is often beneficial when complexity is high, when the project incurs frequent changes, or when the scope is subject to various stakeholders' views of the final software product. Iterative life cycles may include varying degrees of iteration within the spectrum of software development activities. Some of the iterations may include phases that involve only one development stage, whereas others may involve multiple stages, as shown in Figure 2-3.

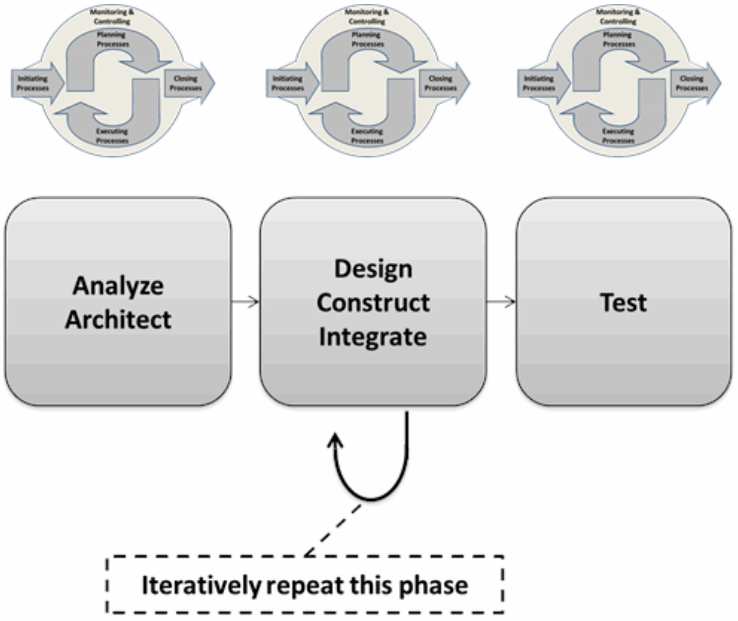


Figure 2-3. An Iterative Project Life Cycle with a Phase having Multiple Stages

- 0**Incremental project life cycle phases.** Incremental project life cycle phases add new increments of functionality that increase the product scope. This approach provides project managers and stakeholders the opportunity to view intermediate demonstrations of working software as meaningful checkpoints. A strictly incremental life cycle produces the increments sequentially. The scope of increments may vary, provided new functionality is

delivered at the end of each incremental phase. The duration of incremental project phases varies widely. Some projects define fewer phases to be completed over a longer time frame, while others define more phases, each having a shorter duration.

- 0**Iterative-incremental project life cycles.** As stated in the Section 2.4.2.3 of the *PMBOK® Guide*, most life cycles develop the product both iteratively and incrementally. This is true of software projects that use incremental life cycle models. Iterative-incremental life cycles occupy a middle position between predictive and adaptive software project life cycles, as illustrated in Figure 2-1.

In addition, the *PMBOK® Guide* states the project scope is generally determined early in the project life cycle, but time and cost estimates are routinely modified as the project team understanding of the product increases. For software projects, requirements may be modified in addition to modifications of time and cost estimates, thus making possible tradeoffs among requirements, time, and cost. As stated in Section 2.4.2.3, one or more of these three factors may be highly constrained, thus limiting the tradeoff options.

An iterative-incremental software project life cycle model is illustrated in Figure 2-4. The prioritized feature set has been partitioned into three iterative-incremental feature sets to be built, where the prioritization of features is based on predetermined prioritization criteria. Prioritizing of the feature sets was preceded by analysis and design phases.

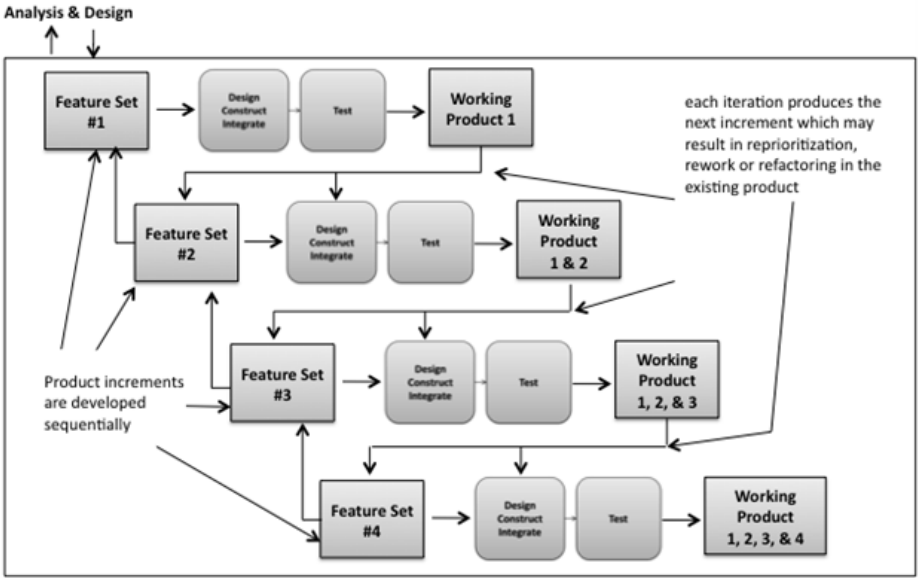


Figure 2-4. Illustrating an Iterative-Incremental Software Project Life Cycle Model

As illustrated in Figure 2-4, increments 2, 3, and 4 produce additional features that build on previously implemented features in accordance with prioritization of the feature sets. The feedback arrows on the left side of the figure indicate that implemented features may have to be reworked based on evolving requirements, issues encountered during implementation, defects discovered when implementing subsequent features, or demonstrations of the working product. Each increment of product capability is verified with respect to the requirements for that feature set and validated by demonstrating incremental capabilities for the appropriate stakeholders.

The time boxes for iterations may vary but iterative-incremental life cycle models for software projects typically uses durations of 1 week to 1 month as the time box for most iterations; some time boxes may be extended to address specific issues. The feedback

arrows from time boxes to the previous ones in Figure 2-4 indicate that adding additional features may expose defects in previous work and that refactoring may be needed to better accommodate the features being added.

Iterative-incremental life cycles for software projects may be on the predictive side of the life cycle continuum or on the adaptive end of the life cycle continuum, depending on the manner in which the prioritized feature sets. A predictive model establishes the feature sets and the priorities among them prior to starting the iterative-incremental development cycles, perhaps to satisfy schedule and resource constraints when the requirements are known and are stable.

An adaptive approach allows feature sets designated for future iterations of the software project life cycle to be reprioritized and modified prior to start of the iterative cycle in which they will be implemented. The number of iterations may be extended as needed or desired.

2.4.2.4 Adaptive Software Project Life Cycles

The considerations presented in Section 2.3.2.4 of the *PMBOK® Guide* are applicable to adaptive software project life cycles. Different adaptive software project life cycle models (on the right side of Figure 2-1) incorporate various elements of software development agility, as follows:

- 0Iterative software development cycles that produce working deliverable software; the duration of an iterative cycle varies from daily to weekly to monthly, but usually not more than monthly;
- 0Continuous, ongoing involvement of a representative customer or user, where customer involvement may occur on a daily basis or during periodic demonstrations of working, deliverable software on a weekly, biweekly, or monthly basis;
- 0Small development teams (e.g., 10 or fewer members) with all team members assigned to one project at a time; large projects include multiple small teams;
- 0Requirement and design may be initially defined or may emerge as the project evolves; in both cases product elements (requirements, design, code) evolve as the project evolves; and
- 0Adaptive life cycles are iterative and incremental. Some typical attributes of iterative software development, which may be incorporated into an adaptive software project life cycle, are presented in Table 2-1.

Table 2-1. Attributes of Iterations in Adaptive Software Project Life Cycles

Attribute¤	Goal¤
Multi-process·Iterations:·Each·iteration·involves·as·many·software·engineering·processes·as·desired·(from·Analyze·to·Test).¤	Permits·extension·of·product·scope·in·a·systematic·manner¶ Frequent·integration·and·test·followed·by·demonstration·yields·higher·quality·software¤
Vertical·Increments:·Iterations·can·deliver·increments·of·functionality·that·involves·as·many·architecture·components·as·desired¤	Integration·risks·and·interface·defects·are·detected·earlier¶ Intermediate·software·becomes·more·understandable·for·the·software·developers·and·from·perspective·of·users·and·stakeholders¤
Short·Duration:·Iterations·typically·range·in·duration·from·1·to·4·weeks¤	Frequent·project·demonstration·and·review·checkpoints¶ Fine-grained·version·control¤
Time-boxed:·Iterations·are·planned·to·all·have·the·same·consistent·duration¤	Simplifies·project·planning¶ Improves·estimates,·based·on·accumulated·data·such·as·velocity·and·burn-down·rate¤

The short duration iterations allows rework to be integrated within the iterations rather than accumulated as a large rework effort that must be accomplished at the end of software development. Performing rework in small increments is more cost effective than the large amount of rework that typically occurs during the integration and testing phases of a predictive life cycle for a software project.

Adaptive project life cycles are particularly appropriate when a precise, early definition of customer needs is difficult or when the technology is used in a way that is different than has historically been applied. An adaptive approach is less expensive than delivering inadequate technical quality or failing to meet the customer's needs. Although adaptive practices improve overall quality and reduce total cost of ownership of the software product over its life cycle, the cost-of-quality curve differs from that of traditional, predictive software project life cycles. This point is discussed further in Section 8 on Software Project Quality Management.

Another important aspect of adaptive software project life cycles is the relationships among product scope, size, cost, and schedule. For many adaptive life cycle projects the cost and schedule for each iterative cycle are fixed because the number of personnel is fixed and the time box for each iterative cycle is the same. The scope of work, and hence the product features that can be implemented (and the resulting amount of software that can be generated) on each iteration is adjusted to fit the constraints of fixed cost and fixed schedule per iteration.

Product size for adaptive software projects is often measured in stories, use cases, or features implemented rather than modules of lines of code implemented. An adaptive project team quickly learns by experience how much work can be accomplished during each iterative cycle. Experience also allows teams to accurately forecast how much time it will take to complete the implementation of a set of features. A measure of productivity call *velocity*, which is the ratio of work products produced to amount of effort expended during an iterative cycle, can be accumulated during development iterations and used to track planned versus actual progress and to forecast final cost and completion date, similar to the way traditional earned value is used to track predictive life cycle projects.

- 0**Intermediate-Adaptive Software Project Life Cycles.** As the name implies, Intermediate-adaptive software project life cycles occupy the region of the life cycle continuum that lies between iterative-incremental and highly adaptive (see Figure 2-1). An illustration of an intermediate-adaptive software project life cycle is provided in Figure 2-5.

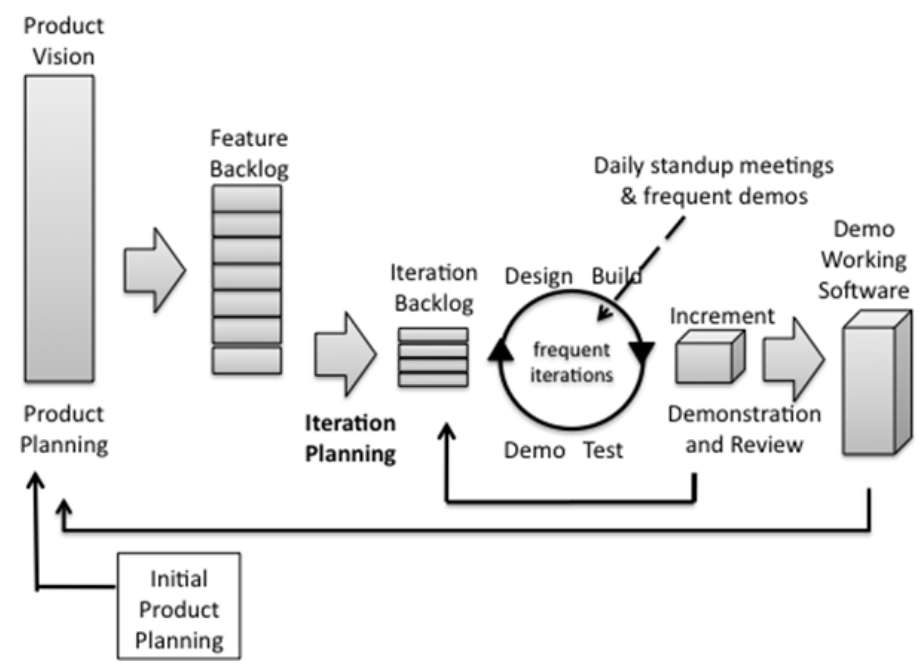


Figure 2-5. An Intermediate-Adaptive Software Project Life Cycle

Key elements of the intermediate-adaptive software project life cycle include the feature backlog and the iteration backlog. Each iterative cycle is based on a set of features and other requirements selected from the feature backlog; this forms the iteration backlog. During the iteration cycle, no changes to the features in the iteration backlog are allowed. Features and other requirements in the feature backlog can be added, deleted, modified, and reprioritized at will. An iteration cycle is typically time-boxed to be one to a few weeks in duration; each cycle may have the same duration or the durations may vary from cycle to cycle but the time box is all iterations is usually fixed. The frequency of internal iterations within an iteration cycle provides the developers with continuing demonstrations of progress; these internal iterations may occur hourly, daily, or weekly, as desired. Daily stand-up meetings are short meetings in which the project team reviews progress, problems, and issues, and agrees on work tasks for that day. A typical approach that might be used internally by the software developers is illustrated in Figure 2-6.

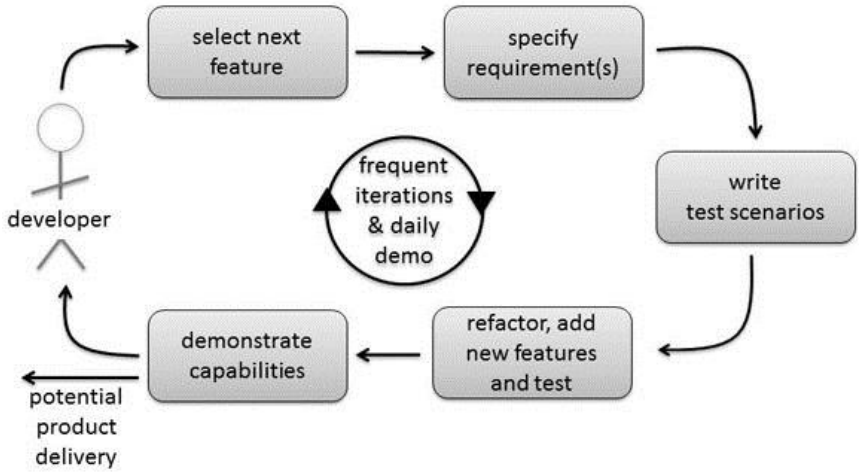


Figure 2-6. Internal Development Cycles for Intermediate-Adaptive Life Cycle

Refactoring may be applied to restructure the existing product base to better accommodate the new feature(s). Some software developers alter the sequence "refactor, add new features, and test" to "test, add new features, and refactor." The latter sequence indicates an approach that iteratively applies the tests to the code, which will fail without the new features, iteratively writes code for the new features and tests until the new code passes the tests, and then refactors the code.

The working software that now includes the new feature(s) is demonstrated. The demonstration may result in accepting the software as demonstrated or may result in revisions. Corrections, additions, and adjustment to the software are easily accommodated because the iteration cycles are short and the functionality that is added each day is small.

It should also be noted that the scope of an adaptive software project life cycle includes other elements of project scope, as appropriate to the needs of the project, such as initial architectural design, independent verification and validation, configuration management, and quality assurance and quality control.

- 0**Highly Adaptive Software Project Life Cycles.** Figure 2-7 illustrates a highly adaptive software project life cycle model that produces daily demonstrations of working software for a knowledgeable customer. The customer is involved on a continuing, daily basis during development of the software product. The customer relates a user story or scenario for a desired feature of the software. Software developers specify product requirements and write test scenarios for implementation of the desired feature or features. The new feature(s) are added, and the test scenarios are applied.

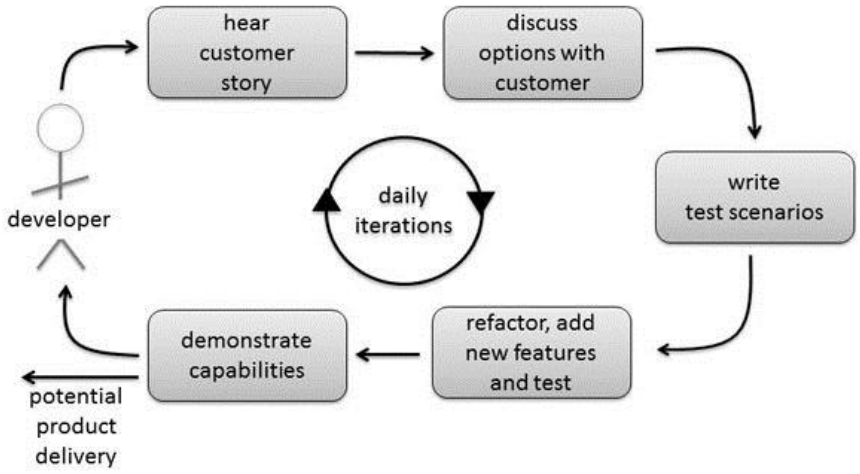


Figure 2-7. Illustrating a Highly Adaptive Software Project Life Cycle

The distinctions between Figure 2-6 and Figure 2-7 are as follows: the customer is "in the loop" on a daily basis in Figure 2-7; the developers are in the loop in Figure 2-6. Daily iterations in Figure 2-7 produce demonstrated capabilities for the customer; daily iterations produce demonstrations for the developers in Figure 2-6. The customer supplies the product requirements for the next iterative cycle in Figure 2-7; the developers select the requirements from the iteration backlog in Figure 2-6. In both cases, each iterative cycle produces working software; the working software must be in deliverable form in both cases.

In Figure 2-7, the working software that now includes the new feature(s) is demonstrated to the customer, who may accept it as demonstrated, may request revisions, or may reject the added features because of miscommunication with the software developers. As stated above, corrections, additions, and adjustment to the software are easily accommodated because the iteration cycles are short and the functionality added each day is small.

Because daily iteration cycles in Figure 2-7 produce working deliverable software, the option exists to deliver the software product into the user environment at any point in time. It should also be noted that the customer determines the features to be included and therefore decides, from the business viewpoint, the value-added expenditure of development time, effort, resources, and money. The overall schedule and budget may be fixed or may grow or shrink adaptively, based on value-added considerations of continuing or terminating product development.

3 PROJECT MANAGEMENT PROCESSES FOR A SOFTWARE PROJECT

This section of the *Software Extension to the PMBOK® Guide – Fifth Edition* describes the ways in which the project management processes in Section 3 of the *PMBOK® Guide* apply to the management of software projects, plus the adaptations and extensions that are commonly used when managing software projects.

The introduction to Section 3 of the *PMBOK® Guide* makes the distinction between project management processes and product-oriented processes as follows:

- 0**Project Management Processes.** These processes ensure the effective flow of the project throughout its existence. These processes encompass the tools and technique involved in applying the skills and capabilities described in the Knowledge Areas (Sections 4 through 13).
- 0**Product-Oriented Processes**. These processes specify and create the project's product. Product-oriented processes are typically defined in the software project life cycle (as discussed in Section 2.4) and vary by application area as well as the stage of the product life cycle.

Sections 4 through 13 of this software extension are based on the corresponding sections of *PMBOK® Guide*; they present the tools and techniques that are typically used to manage software projects. Section 2.4 of this software extension describes project life cycles used to manage the various stages of the software product life cycle.

The project-oriented (project management) processes and the product-oriented processes for software projects are closely aligned in the iterative-incremental to highly adaptive life cycles part of the life cycle continuum for software projects presented in Section 2.4 of this extension.

As noted in Section 3 of the *PMBOK® Guide,* it should be used as a guide in managing a project while considering the overall approach to be followed for the project. This effort is known as tailoring. Section 2.4 and Sections 4 through 13 of this extension provide tailoring guidelines for the processes in the *PMBOK® Guide* that are used to manage a software project.

This section of this software extension presents extensions to, and adaptations of, the five Process Groups presented in Section 3 of the *PMBOK® Guide*.

3.1 Common Project Management Process Interactions

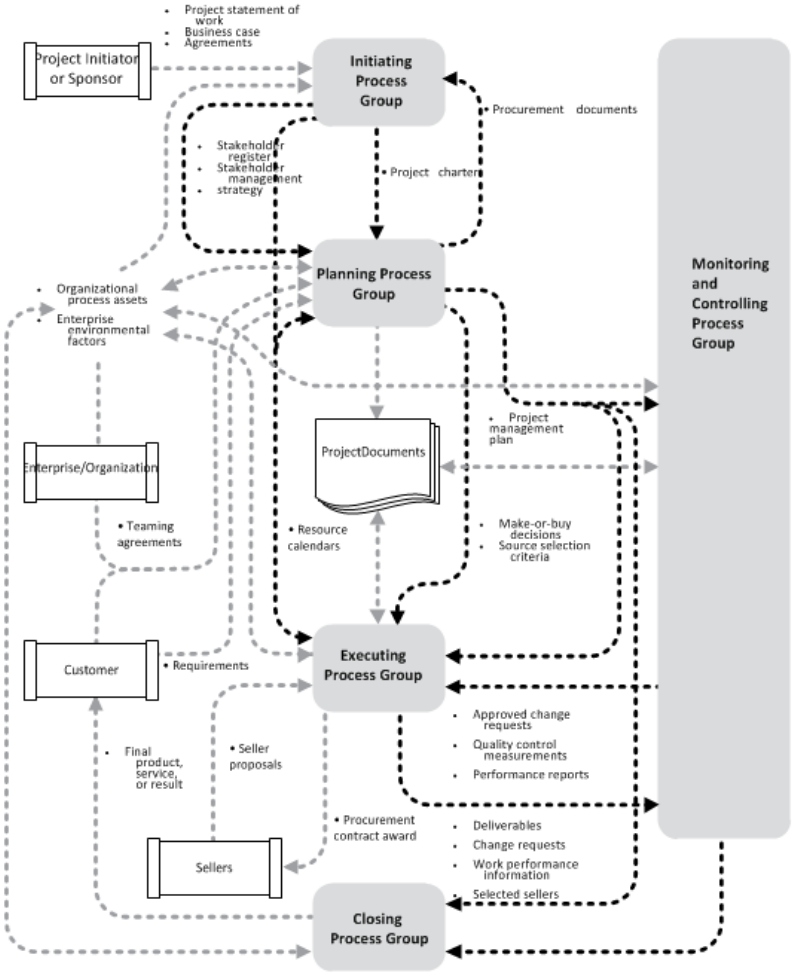
As stated in Section 3.1 of the *PMBOK® Guide*, application of the project management processes is often iterative, with many processes repeated during a project. Section 2.4

of this software extension indicates that software project life cycles commonly occupy a continuum ranging from predictive models on the one end to highly adaptive models at the other end. Variations along this continuum are generally based on the manner in which life cycle elements are applied along the iterative-incremental to highly adaptive life cycle continuum for software projects.

There is an ongoing relationship between the Process Groups and the project phases for software projects. Predictive life cycles include activity-oriented phases (i.e., analysis, design, implementation, etc.). For example, the requirements phase is initiated, the requirements targets are planned, the requirements phase is executed and monitored and controlled, and the requirements phase is closed out; the design phase is initiated and the cycle begins again.

The iterations that occur in the iterative-incremental to highly adaptive life cycles for software projects can be thought of as incremental phases. Each phase (each iteration) includes some degree of initiating, planning, executing, and monitoring and control. Closing out an iteration typically involves the demonstration of working software. The demonstration provides the basis for initiating and planning the next iteration. For software projects, the duration of a Planning and Executing process cycle typically varies from 2 to 4 weeks for an iterative-incremental cycle to as little as one day for a highly adaptive iteration.

Figure 3-3 in the *PMBOK® Guide* illustrates the interactions among the Planning, Executing, and Monitoring and Controlling Process Groups within a project phase. Figure 3-1 is an adaptation of *PMBOK® Guide* Figure 3-3 for software projects.



Note: For software development, the planning, executing, and monitoring and controlling processes require high levels of collaboration among project stakeholders.

Figure 3-1. Interactions Among Process Groups for Software Development

For adaptive software project life cycles, these three Process Groups (Planning, Executing, and Monitoring and Controlling) are often so closely interrelated that they are not distinguishable as separate processes. For example, Planning (entry to an iteration cycle) may involve selecting the next set of features to be implemented, but daily demonstrations of working software code may alter the planned set of features to be implemented within a given iteration cycle. Resources used and progress made provide tangible evidence for detailed Monitoring and Controlling. Exiting an adaptive development cycle involves demonstrating working software code to a customer or a designated user representative and obtaining their acceptance of the implemented functionality or accepting their requests for further changes. Figure 3-2 illustrates the interactions and overlaps where each "hump" is an iterative development cycle; each cycle involves the *PMBOK® Guide* process groups of planning, executing, and monitoring and controlling.

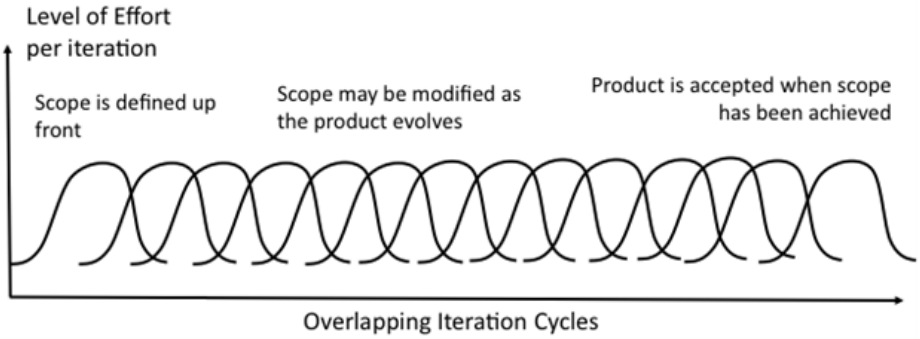


Figure 3-2. Overlap of the Planning, Executing, and Monitoring and Controlling Processes

The overlap of phases for adaptive software life cycles indicates that team members need to have all of the necessary skills to plan, execute, and monitor and control each iteration of an adaptive software project because all skills are constantly needed and must be available within the team, rather than relying on other elements of the organization to provide the needed skills.

3.2 Software Project Management Process Groups

According to the *PMBOK® Guide,* these five Process Groups have clear dependencies and are typically performed in the same sequence on each project. In contrast, and as indicated in Section 2.4 of this software extension and the above discussion, the ways in which the five *PMBOK® Guide* Process Groups are applied to software projects may vary from project to project and within the iterative cycles of a software project.

Software development projects based on adaptive life cycles involve frequent and closely coordinated interactions between the customer and the project's processes—particularly in the translation of customer requirements into planning, which gets communicated to execution. Of particular importance, the process flow is not strictly one-directional feed-forward, where information is fed sequentially from one process to the next. In software development, there is a need for frequent feedback between planning and executing and monitoring and controlling to ensure that the emerging software product is consistent with (possibly changing) expectations. Documentation of decisions made is necessary; but documentation alone is not sufficient to provide the understanding needed to implement a software product that meets the needs of the customer and the business. Frequent interpersonal interactions are required to provide clarity for all stakeholders; therefore, the emphasis is placed on evolving, working software for each development cycle of an adaptive software project life cycle, in addition to documentation.

It is also important to recognize that the life cycle continuum for software projects is not a thin line; it is multi-dimensional. All of the processes and support functions for software development (configuration management, quality assurance, documentation, and so forth) should be adapted to fit the needs of each software project life cycle.

3.3 Initiating Process Group

As stated in the *PMBOK® Guide*, internal and external stakeholders who will interact and influence the overall outcome of the project are identified. One of the most important stakeholders for a successful software project is a knowledgeable customer or designated user representative who can state the users' wants and needs on a continuing, ongoing basis. Identifying such a stakeholder early will allow for frequent demonstrations of the incrementally evolving product and the associated feedback will ensure that the right features are delivered. It is also important to discuss issues with experienced project managers and technical leaders on similar projects, or perhaps managers and leaders on releases of previous versions of a software product undergoing significant modifications.

3.4 Planning Process Group

According to the *PMBOK® Guide,* The Planning Process Group consists of those processes performed to establish the total scope of the effort, define and refine the objectives, and develop the course of action required to attain those objectives.

These processes are often overlapped, interleaved, and iterated in various ways for software projects. Section 5 of this software extension (Scope), explains that the scope of a software project, the objectives to be obtained, and the courses of action to be followed when conducting a software project are often adjusted as the project evolves. Adaptive software project life cycles are designed to encourage frequent customer interactions and to permit graceful incorporation of changes throughout the project life cycle.

3.5 Executing Process Group

According to the *PMBOK® Guide*: During project execution, results may require planning

updates and re-baselining. This may include changes to expected activity durations, changes in resource productivity and availability, and unanticipated risks.

Every software project is a unique undertaking and is, therefore, a learning experience. Changes in activity durations, resource productivity and availability, and unanticipated risks are the norm for many software projects. Anticipating and facilitating these changes is a primary goal of adaptive software project lifecycles.

3.6 Monitoring and Controlling Process Group

Depending on the software project life cycle used, monitoring and controlling of software projects can vary from traditional techniques for predictive software project life cycles (i.e., preplanned milestones, earned value tracking, and technical performance measurement) to reliance on frequent demonstrations of progress (or lack of progress) for adaptive software project life cycles. These iterative demonstrations vary from daily demonstrations for the project team to weekly or monthly demonstrations for customers and users.

3.7 Closing Process Group

The techniques presented in the Closing Process Group for a software project, as specified in the *PMBOK® Guide,* are equally applicable to software projects. Demonstration of working software is an important approach to closing a predictive life cycle project or a project phase (i.e., an iterative cycle) for software projects that use life cycles on the iterative-incremental to highly predictive side of the life cycle continuum presented in Section 2.4 of this software extension. In all cases, it is important to conduct a retrospective, lessons-learned session; assess team performance; and update the organizational knowledge base.

3.8 Project Information

Project information collected during a software project can be used to predict attributes such as final cost and delivery date (for predictive software project life cycles) and incremental progress using measures such as velocity and burn down rate (for adaptive software lifecycles) (see the Glossary for definitions of these terms and Section 7 for additional information). Project information can be collected and saved in an organizational database to provide a basis for estimating future software projects that have similar characteristics (i.e., similar domains, customers, software developers, development tools). Care should be taken to ensure that the attributes of past and future projects are sufficiently similar when using past performance data to estimate a future project.

3.9 Role of the Knowledge Areas

As indicated in Section 3.9 of the *PMBOK® Guide*, the 47 project management processes identified in the *PMBOK® Guide* are further organized into ten separate Knowledge Areas (note PMI-specific use of the term *processes,* which is used in different contexts). These 10 Knowledge Areas are presented in Sections 4 through 13 of the *PMBOK® Guide* and describe the tools and techniques that are used on most projects most of the time. In a similar manner, the 10 Knowledge Areas presented in Sections 4 through 13 of this software extension describe the tools and techniques that are used for most software projects, most of the time.

4 SOFTWARE PROJECT INTEGRATION

MANAGEMENT

As stated in the introduction to Section 4 of the *PMBOK® Guide:* "Project Integration Management includes the processes and activities needed to identify, define, combine, unify, and coordinate the various processes and project management activities within the Project Management Process Groups." Many of the inputs, tools and techniques, and outputs in Section 4 of the *PMBOK® Guide* are applicable to integration management of software projects. This section of the *Software Extension to the PMBOK® Guide – Fifth Edition* presents extensions that are especially important for software project integration management.

It is important to note that "software project integration management" refers to the integration of the processes and activities in the *PMBOK® Guide* project Knowledge Areas (Sections 4 through 13); it does not refer to the software project life cycle process of integrating software components to form a partial or finished software product.

Planning and conducting a software project is mostly a proactive endeavor, rather than integration and coordination of subsidiary plans, as presented in Section 4 of the *PMBOK® Guide*. Sometimes, other departments provide some software project functions (e.g., configuration management, independent testing) but, for the most part, software project managers are responsible for planning and conducting a wide scope of project activities (see Section 5).

Software projects have a wide range of influences that impact the rigor of and emphasis on various project management activities. There is no single way best to manage a software project. Every project management process needs to be addressed to determine the appropriate level of implementation for each project endeavor. Project managers should apply knowledge and skills appropriate to software projects by tailoring and adapting the project management processes in the *PMBOK® Guide* and this software extension to maximize the potential of the project team to achieve the desired project performance.

Figure 4-1 provides an overview of Software Project Integration Management; it is an adaptation of Figure 4-1 in the *PMBOK® Guide*.

Figure 4-1. Software Project Integration Management Overview

4.1 Develop Software Project Charter

According to the *PMBOK® Guide*, Section 4.1: "Develop Project Charter is the process of developing a document that formally authorizes the existence of a project and provides the project manager with the authority to apply organizational resources to project activities." The inputs, tools and techniques, and outputs presented in Section 4.1 of the *PMBOK® Guide* are applicable to developing a software project charter.

4.1.1 Develop Software Project Charter: Inputs

The inputs in Section 4.1.1 of the *PMBOK® Guide* are applicable for software projects. An additional consideration is the role played by a software project portfolio when developing a software project charter (see 4.1.1.6 of this extension).

4.1.1.1 Project Statement of Work

See Section 4.1.1.1 of the PMBOK® Guide.

4.1.1.2 Business Case

See Section 4.1.1.2 of the PMBOK® Guide.

4.1.1.3 Agreements

See Section 4.1.1.3 of the PMBOK® Guide.

4.1.1.4 Enterprise Environmental Factors

See Section 4.1.1.4 of the PMBOK® Guide.

4.1.1.5 Organizational Process Assets

See Section 4.1.1.5 of the PMBOK® Guide.

4.1.1.6 Software Project Portfolios

Supporting business objectives are the primary reasons that organizations invest in IT and software projects. Contracted projects for external customers provide jobs and sources of revenue; internal projects support the mission and business strategy of the organization. Many organizations maintain a portfolio of software projects and follow a portfolio weighting and planning process for selecting software projects that will align with business strategy. Within a portfolio, investments in projects are linked to strategic objectives, identification of potential benefits, and order of magnitude estimates for completing projects, including the types and amounts of resources needed. Additional weighting criteria include the number of resources needed (software developers, other software personnel, technology), the timeframe within which a product is needed, and priority of need.

Additional aspect of software portfolio management include identifying specific goals for each project by developing a preliminary scope statement along with benefits and constraints, group and prioritize related potential projects based on capacity, prepare a charter for the next most important project to maximize benefit, and coordinate projects to make the best use of delivery capability.

The result is a prioritized backlog of software projects mapped to the organization's strategic plan and availability of resources. The relationship between software projects and software portfolio management were presented in Section 1 of this software extension and illustrated in Figure 1-1.

4.1.2 Develop Software Project Charter: Tools and Techniques

The tools and techniques presented in Section 4.1.2 of the *PMBOK® Guide* are applicable for developing a software project charter with the following additional information.

As discussed in Section 2, there is continuum of software project life cycles ranging from predictive, where the project scope and life cycle tools and techniques are planned during project initiation and planning; to iterative-incremental, where the project is planned to generate successive increments of increasing functionality on each iteration; to adaptive, where the product scope emerges during iteration cycles; Charters for software projects that use predictive and iterative-incremental life cycles on the predictive side of the

continuum provide clear and definitive statements of desired outcomes. Charters for adaptive projects state a clear vision or business goal and acceptable project constraints under which to pursue that vision or business goal. The selected life cycle model influences project planning, execution, and monitoring and controlling of each of the project management processes.

Domain experts should be consulted when developing a software project charter. Expertise in developing similar systems using similar development platforms, systems software, product architecture, and information design (i.e., databases, data interchanges, and data warehouses) can provide valuable insights and expose unrecognized complexities and risk factors. In addition, when software projects involve working with an existing implementation or team, inputs from experts familiar with the architecture, technical implementation, testing approach, and team capabilities can provide assistance in developing the project charter as well.

4.1.2 Develop Software Project Charter: Tools and Techniques

4.1.2.1 Expert Judgment

See Section 4.1.2.1 of the *PMBOK® Guide*.

4.1.2.2 Facilitation Techniques

See Section 4.1.2.2 of the *PMBOK® Guide*.

4.1.3 Develop Project Charter: Outputs

See Section 4.1.3 of the *PMBOK® Guide,* with the addition of 4.1.3.2.

4.1.3.1 Software Project Charter

See Section 4.1.3.1 of the PMBOK® Guide.

4.1.3.2 Updates to the Organization's Software Project Portfolio

An additional consideration when developing a software project charter is updating of the software project portfolio, if there is one, with the information developed during preparation of the software project charter.

4.2 Develop Software Project Management Plan

According to Section 4.2 of the *PMBOK® Guide*, Develop Project Management Plan is the process of defining, preparing, and coordinating all subsidiary plans and integrating them into a comprehensive project management plan.

The extent to which planning software projects involves "defining, preparing, and coordinating all subsidiary plans and integrating them" depends on the software project life cycle selected, the organizational structure, and the project context. The software project manager's job may not be to perform all project planning activities—some may be performed by a software project management office (such as preparing estimates based on historical data) or by another organizational unit (such as independent testing). Project Integration Management ensures that all necessary processes are being performed with sufficient rigor and that sufficient project performance information will be produced so that the software project manager can perform the appropriate Monitor and Control

activities.

As discussed in Section 2 of this software extension, predictive software project life cycles typically involve more upfront planning than do adaptive life cycles. The pros and cons of each approach as well as the intermediate iterative-incremental approaches in the life cycle continuum are presented in Section 2.4.2 of this extension.

Developing a software project plan for a predictive life cycle project may involve upfront planning specialists and other elements of the organization in process areas such as configuration management and quality assurance to a greater extent than when planning an adaptive software project. Regardless of the life cycle adopted, software projects may also need to integrate a number of additional plans such as an information security plan, data management plan, product launch plan, deployment plan, and perhaps a team training plan for new technologies or for a new domain.

Project managers of predictive life cycle software projects tend to put substantial effort into upfront development of the project plan and integration of subsidiary plans developed by support personnel from other organizational units (e.g., estimation specialists in the PMO). In adaptive projects, there is less upfront effort spent on developing detailed scope, cost, and schedule plans, but significant effort is typically spent on defining monitor and control processes, such as requirements traceability, to ensure coordination among the project members or project teams as the emerging plans are implemented.

Project constraints that influence development of a software project plan include organizational policies, the size and criticality of the project, risk mitigation strategies, and the complexity of the solution. Projects with rigid constraints may require increased emphasis on the Risk Management, Integration Management, Procurement Management, Quality Management, and Stakeholder Management processes as described in the corresponding Knowledge Areas.

Because software is developed by the coordinated intellectual effort of the team members it is not advisable to plan a large team for a large software project. A better approach is to scale up by increasing the number of teams, which limits the number of communication paths within each team. Small teams (ten or fewer members) can develop sub-elements that have well-defined interfaces to the other product elements; the project plan can include the integration process, which typically occurs following software construction for predictive life cycles and is integrated into adaptive life cycles. Software projects that have multiple teams typically have team leaders who report to the project manager. The team leaders are technical contributors who also coordinate the team's work.

Large complex projects may be organized as multiple subprojects with a project management team, or multiple distinct projects with a program manager to coordinate the multiple projects, each of which will have a project manager and team leaders within each project.

In these cases it is important to allocate requirements and interfaces to project, subproject, and team so that the development of product components can proceed concurrently with a plan for periodic integration of work products. There will be more emphasis on Human Resources Management and Communications Management processes as the scale of a project grows (see Sections 10 and 13).

The form of a software project management plan and the emphasis placed on various aspects of the plan depends on many factors, including but not limited to the project and product scope, product requirements, the choice of software project life cycle model, the organizational assets, the influence of contextual factors, and the nature of the customer relationship.

For example, the choice of a predictive or an adaptive project life cycle model will influence the planning for management of scope, time, cost, and product integration, among others. A project that will involve a geographically distributed or virtual project team will place emphasis on human resource issues and management of stakeholders. Perceived complexity of the product and familiarity of the software team with the problem domain and the technology to be used will place emphasis on planning for quality control, quality

assurance, and risk management. Dealing with vendors and subcontractors will place emphasis on planning for procurement activities.

The Knowledge Areas in the *PMBOK® Guide* and this software extension provide guidance on dealing with these issues when preparing a software project management plan.

Regardless of the software project life cycle adopted, the development and integration of elements of a software project management plan is rarely a linear process as elements evolve at different rates and exert different levels of influence on each element. Conducting a software project is a learning process that results in revision of the project plan and subsidiary plans as understanding grows; re-planning of both the project and product is inevitable for software projects, regardless of the life cycle used.

4.2.1 Develop Software Project Management Plan: Inputs

Figure 4-5 and Section 4.2.1 of the *PMBOK® Guide* indicate that developing a project management plan involves integration of related and subordinate plans that have been developed. Development of a software project plan tends to be a process of planning the primary activities and coordinating subordinate plans rather than integration of plans developed by others.

Estimates of cost, schedule, technical infrastructure, and risk provide important inputs for developing a project management plan, as indicated in Section 4.2.1 of the *PMBOK® Guide*. Every software project differs from all past projects (because software replication is a trivial process) so there are typically many unknowns and uncertainties during the initiating and planning stages of a software project, which results in imprecise and sometimes inaccurate software project estimates. Estimation techniques include analogies, expert judgment (perhaps involving a Delphi process), rules of thumb, statistical techniques such as PERT, and estimation algorithms that rely on historical data for calibration to the local situation.

Many software project managers use two or more estimation techniques to expose differences in the assumptions and reasoning processes used to arrive at the (often disparate) estimates. Templates can be tailored and adapted to fit the needs of different software projects; they can guide the thought processes and work activities of a software project manager when preparing a project management plan.

ISO/IEC/IEEE Standard 16326 – Systems and software engineering – Life cycle process – Project Management **[15]** provides useful input information for planning software projects.

4.2.1.1 Project Charter

See Section 4.2.1.1 of the PMBOK® Guide.

4.2.1.2 Outputs from Other Processes

Outputs from the processes described in Sections 5 through 13 of the *PMBOK® Guide* and this software extension are integrated to create the project management plan.

4.2.1.3 Enterprise Environmental Factors for Software Projects

Environmental factors that may impact planning a software project include existing technical assets such as software that can be reused, development and testing infrastructure and facilities, and the technical infrastructure, including networks, data centers, and simulation and modeling facilities.

4.2.1.4 Organizational Process Assets for Software Projects

Organizational process assets listed in Section 4.2.1.4 of the *PMBOK® Guide* are applicable for software projects. In addition, change control and configuration management method and tools for software projects are needed to control software code baselines because software code is updated and changed frequently during software development; it is not uncommon for software code to be changed several times in one day when using an adaptive life cycle.

Without effective version control, the incorrect version of a software module may be used as the basis for further work or may be incorrectly integrated into a deliverable product baseline.

Some product changes have local impact and other changes should be reviewed and approved by a software change control board, depending on the wider impact of a proposed change. Predictive life cycles tend to exert tighter control over the software code baselines during software development than occurs for adaptive software project life cycles because, when using an adaptive life cycle, the frequent iterations and demonstrations of working, deliverable code will expose undesired side effects of changes that can be quickly corrected. Frequent, ongoing changes are typically more difficult when using a predictive software project life cycle model.

4.2.2 Develop Software Project Management Plan: Tools and Techniques

The tools and techniques in Section 4.2.2 of the *PMBOK® Guide* are applicable to development of a software project management plan. Templates and estimation tools are useful when developing a software project plan.

4.2.2.1 Expert Judgment

See Section 4.2.2.1 of the *PMBOK® Guide.*

4.2.2.2 Facilitation Techniques

See Section 4.2.2.2 of the *PMBOK® Guide*.

4.2.3 Develop Software Project Management Plan: Outputs

4.2.3.1 Project Management Plan

In addition to the project management plan listed as the output in Section 4.2.3.1 of the *PMBOK® Guide*, the following software specific documents and plans may be included as supporting output documents:

- 0Requirements management plan,
- 0Configuration management plan,
- 0Security plans (physical, project, data),
- 0Enterprise technology insertion plan,
- 0Information security plan,
- 0Test and evaluation plan,
- 0Data management plan,
- 0Deployment plan,
- 0Technology infrastructure plan, and
- 0Training plan for the software team.

In addition, deployment plans and technology infrastructure plans may be developed for IT infrastructure projects and for software projects that involve deployment of the software product to external customer sites.

4.3 Direct and Manage Software Project Work

Software project managers who manage predictive life cycle projects tend to follow more closely the traditional approaches to directing and managing project work as indicated in Section 4.3 of the *PMBOK® Guide* than do managers of adaptive software project life cycle projects. A desirable attribute of managing co-located multifunctional software teams is that they be self-enabled and self-directed, which places the project manager in a more hands-off position in the day-to-day management of the project team than the manager of a predictive life cycle project team. However, this does not mean the software project manager is left without a role in directing and managing adaptive software projects. Activities engaged in by project managers of adaptive life cycle software projects include, but are not limited to:

- 0Communicating the project scope, resources, and schedule/budget constraints to the project team and other appropriate stakeholders;
- 0Providing the resources and facilities needed by the collective software team;
- 0Monitoring and controlling project resources, schedule, and budget;
- 0Controlling changes to project scope, resources, schedule, and budget;
- 0Consulting with and allocating the work among multiple software development teams;
- 0Ensuring communication and coordination of work activities among multiple teams and coordinating the integration of resources and components;
- 0Facilitating ongoing and continuous communications between the customer/user representatives and the software developers;
- 0Monitoring productivity, product quality, and team performance while making adjustments as needed;
- 0Managing risk;
- 0Ensuring effective interactions with other organizational units, such as independent testing and user training;
- 0Facilitating demonstrations of evolving product capabilities for appropriate stakeholders;
- 0Facilitating delivery of early product capabilities into the user environment, as desired;
- 0Facilitating delivery and acceptance of the final product and closing the project; and
- 0Coordinating dependencies with related projects, which may be elements of a common development program or milestone dependencies with related IT infrastructure projects.

Managers of all software projects engage in these activities, whether using a predictive, iterative-incremental, or adaptive life cycle although the details vary with the life cycle model used.

4.3.1 Direct and Manage Software Project Execution: Inputs

See Section 4.3.1 of the *PMBOK® Guide* for inputs that are generally applicable to directing and managing software project execution.

4.3.1.1 Project Management Plan

See Section 4.3.1.1 of the *PMBOK® Guide.*

4.3.1.2 Approved Change Requests

See Section 4.2.1.2 of the PMBOK® Guide.

4.3.1.3 Enterprise Environmental Factors

See Section 4.3.1.3 of the *PMBOK® Guide*.

4.3.2 Direct and Manage Software Project Execution: Tools and Techniques

The tools and techniques presented in Section 4.3.2 of the *PMBOK® Guide* are generally applicable to directing and managing software project execution. In addition to these, information dissemination is also an important tool/technique for directing and managing software project execution.

4.3.2.1 Expert Judgment

See Section 4.3.2.1 of the *PMBOK® Guide.*

4.3.2.2 Project Management Information System

See Section 4.3.2.2 of the *PMBOK® Guide.*

4.3.2.3 Meetings

See Section 4.3.2.3 of the *PMBOK® Guide.*

4.3.2.4 Information Dissemination

Because software tends to be an invisible product, as compared to physical artifacts, tools and techniques for disseminating project information are particularly important for software projects. Providing team members, management, customer, external stakeholders, and everyone who affects or is affected by a software project, at the level appropriate for each constituency, is thus an important activity for a software project manager. The kind of information to be disseminated includes, but is not limited to:

- 0Current status of the overall project;
- 0Risks and status (watch list, monitored, confronted);
- 0Current work assignments;
- 0Daily progress and remaining work;
- 0Number of use cases, features, or stories written/implemented/delivered;
- 0Number of test cases and test scenarios written/passed;
- 0Product components/features implemented with cost or staff-hours as the independent variable;
- 0Resolutions and action items from the last retrospective meeting; and
- 0Status of servers and other infrastructure equipment (up, down, in maintenance).

See Section 10 (Software Project Communications Management) for additional information concerning information dissemination.

Some software projects use large displays (i.e., information radiators) posted in locations where software developers and other team members can easily see them; the displays are typically posted in locations where it is difficult to avoid seeing them. The purpose of visual displays is to communicate essential project information that personnel need to know without anyone having to ask questions. This approach facilitates increased communication with less confusion and misinterpretation for the project team and other stakeholders. Well-designed visual displays are:

- 0Large and readily visible,
- 0Understood at a glance, and
- 0Changed periodically and kept up to date.

Typically, a visual display is paper-based and is posted in the team room or in a hallway. Sometimes visual displays are presented on readily accessible Web pages. Visual displays can be used to inform stakeholders outside of the project team concerning project status and other issues that may be of concern to them.

4.3.3 Direct and Manage Software Project Execution: Outputs

The outputs presented in Section 4.3.2 of the *PMBOK® Guide* are generally applicable to directing and managing software project execution. In addition to these, an additional output is provided in Section 4.3.3.6 of this extension.

4.3.3.1 Deliverables

See Section 4.3.3.1 of the *PMBOK® Guide*.

4.3.3.2 Work Performance Data

See Section 4.3.3.2 of the *PMBOK® Guide*.

4.3.3.3 Change Requests

See Section 4.3.3.3 of the *PMBOK® Guide*.

4.3.3.4 Project Management Plan Updates

See Section 4.3.3.4 of the *PMBOK® Guide*.

4.3.3.5 Project Document Updates

See Section 4.3.3.1 of the *PMBOK® Guide*.

4.3.3.6 Demonstrations of Working, Deliverable Software

In addition to the outputs contained in See Section 4.3.3.1 of the *PMBOK® Guide*,, continuing and ongoing demonstrations of working, deliverable software are the most important indicators of tangible progress for directing and managing iterative-incremental and adaptive software projects. Productivity and progress indicators such as velocity, burn-down rates, and buildup charts provide the work performance data for adaptive life cycle projects.

4.4 Monitor and Control Software Project Work

The inputs, tools and techniques, and outputs for monitoring and controlling project work indicated in Section 4.4 of the *PMBOK® Guide* are applicable for managing software projects. When managing adaptive software projects, delivered increments of working software code are evaluated against the project constraints, the team performance, and the overall goals of the project to trigger change control events, as necessary, when those events exceed control limits.

Figure 4-2 illustrates how management of portfolios and programs can influence monitoring and controlling of software projects and the backlog of work for project teams.

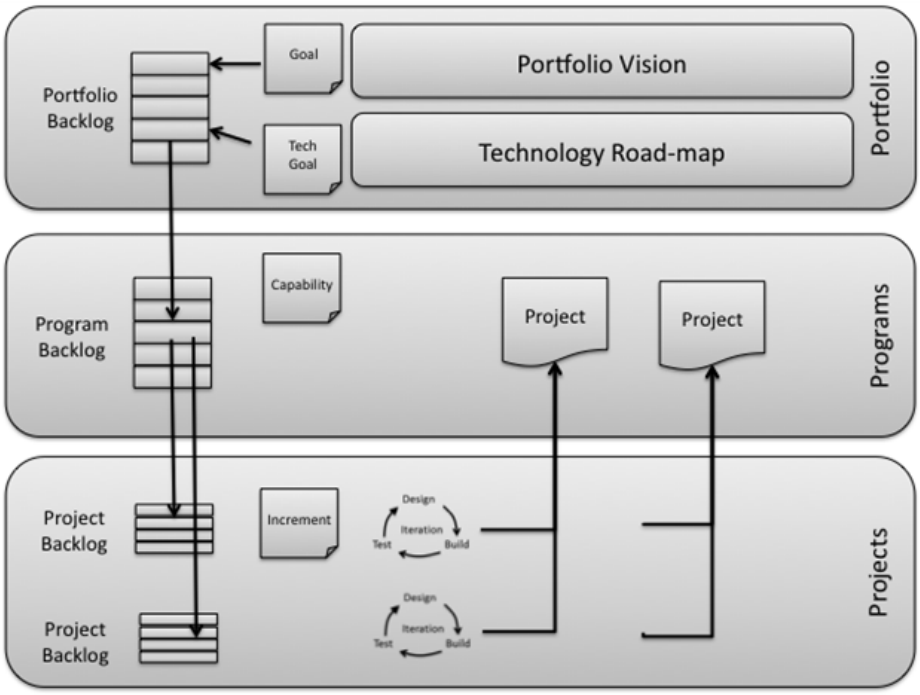


Figure 4-2: Relationships Among Software Portfolios, Programs, and Projects

4.4.1 Monitor and Control Software Project Work: Inputs

The inputs for monitoring and controlling project work in Section 4.4.1 of the *PMBOK® Guide* are applicable for monitoring and controlling software project work.

4.4.1.1 Project Management Plan

See Section 4.4.1.1 of the *PMBOK® Guide.*

4.4.1.2 Schedule Forecasts

See Section 4.4.1.2 of the *PMBOK® Guide*.

4.4.1.3 Cost Forecasts

See Section 4.4.1.3 of the *PMBOK® Guide*.

4.4.1.4 Validated Changes

See Section 4.4.1.4 of the *PMBOK® Guide*.

4.4.1.5 Work Performance Information

See Section 4.4.1.5 of the *PMBOK® Guide*.

4.4.1.6 Enterprise Environmental Factors

See Section 4.4.1.6 of the *PMBOK® Guide*.

4.4.1.7 Organizational Process Assets

See Section 4.4.1.7 of the *PMBOK® Guide.*

4.4.2 Monitor and Control Software Project Work: Tools and Techniques

The tools and techniques for monitoring and controlling project work in Section 4.4.2 of the *PMBOK® Guide* are applicable to monitoring and controlling software projects.

4.4.2.1 Expert Judgment

See Section 4.4.2.1 of the *PMBOK® Guide*.

4.4.2.2 Analytical Techniques

See Section 4.4.2.2 of the *PMBOK® Guide*.

4.4.2.3 Project Management Information System

See Section 4.4.2.3 of the *PMBOK® Guide*.

4.4.2.4 Meetings

See Section 4.4.2.4 of the *PMBOK® Guide*.

4.4.3 Monitor and Control Software Project Work: Outputs

The outputs for monitoring and controlling project work in Section 4.4.3 of the *PMBOK® Guide* are applicable to monitoring and controlling software projects.

4.4.3.1 Change Requests

See Section 4.4.3.1 of the *PMBOK® Guide.*

4.4.3.2 Work Performance Reports

Work performance reports for software project, in addition to those in Section 4.4.3.2 of the *PMBOK® Guide* include:

- 0Earned value reports with projections of estimated actual cost and estimated completion date;
- 0Updates to estimates (product size, delivered quality, delivery date);
- 0Team productivity measures such as velocity metrics, feature burn down and burn up charts;
- 0Feature backlogs; and
- 0Configuration management reports.

4.4.3.3 Project Management Plan Updates

See Section 4.4.3.3 of the *PMBOK® Guide*.

4.4.3.4 Project Documents Updates

See Section 4.4.3.4 of the *PMBOK® Guide*.

4.5 Perform Integrated Software Change Control

See Section 4.5 of the *PMBOK® Guide,* which is applicable for managing software projects, with the following extensions.

4.5.1 Perform Integrated Software Change Control: Inputs

Variations that trigger change control processes differ for different software project life cycles. Control limits for schedule, cost, defects, and product scope are usually established for predictive software project life cycle projects. Exceeding a control limit triggers a change control process. For adaptive project life cycles, change control is not usually required for occasional anomalies, as long as the vision and goals of the project can be achieved within the constraints of schedule and cost.

The inputs for performing integrated change control, as presented in the *PMBOK® Guide,* are applicable for performing integrated software change control.

4.5.1.1 Project Management Plan

See Section 4.5.1.1 of the *PMBOK® Guide*.

4.5.1.2 Work Performance Reports

See Section 4.5.1.2 of the *PMBOK® Guide*.

4.5.1.3 Change Requests

See Section 4.5.1.3 of the *PMBOK® Guide*.

4.5.1.4 Enterprise Environmental Factors

See Section 4.5.1.4 of the *PMBOK® Guide*.

4.5.1.5 Organizational Process Outputs

See Section 4.5.1.4 of the *PMBOK® Guide*.

4.5.2 Perform Integrated Software Change Control: Tools and Techniques

Not all changes to software require formal approval, but all proposed changes should be evaluated for impacts on users and other operational stakeholders. Changing the color of and icon on a user display may not be important, or it may render the application unusable if some users are color blind or where compliance with disability regulations concerning color blindness may be required. Adding an additional choice on a web page may have no impact on present capabilities or it may render a data interchange capability inoperable. Thus, some seemingly minor changes may have significant impacts.

For predictive software life cycle projects, a separate change control process that includes change requests and change control boards is used. For adaptive life cycle projects, a change request is another element of the product backlog; the iteration planning process does not distinguish between new scope, change requests, and defect fixes although the nature of the backlog element may determine the priority of the element within the backlog.

The tools and techniques for performing integrated change control, as presented in the *PMBOK® Guide,* are applicable for performing integrated software change control.

4.5.2.1 Expert Judgment

See Section 4.5.2.1 of the *PMBOK® Guide*.

4.5.2.2 Meetings

See Section 4.5.2.2 of the *PMBOK® Guide*.

4.5.2.3 Change Control Tools

See Section 4.5.2.3 of the *PMBOK® Guide*.

4.5.3 Perform Integrated Software Change Control: Outputs

See Section 4.5.3 of the *PMBOK® Guide* for outputs applicable to performing integrated change control for software projects.

4.5.3.1 Approved Change Requests

See Section 4.5.3.1 of the *PMBOK® Guide.*

4.5.3.2 Change Log

See Section 4.5.3.2 of the *PMBOK® Guide*.

4.5.3.3 Project Management Plan Updates

See Section 4.5.3.3 of the *PMBOK® Guide*.

4.5.3.4 Project Documents Updates

See Section 4.5.3.4 of the *PMBOK® Guide*.

4.6 Close Software Project or Phase

According to Section 4.6 of the *PMBOK® Guide*, Close Project or Phase is the process of finalizing all activities across all five Project Management Process Groups to formally complete the project or phase. The inputs, tools and techniques, and outputs for closing a project, as indicated in Section 4.6 of the *PMBOK® Guide* are applicable to closing a software project.

Two items are particularly important when closing a software project: historical data and lessons learned. This information is captured in an organizational data repository or repositories. Historical data provides a basis for estimating future, similar projects. Historical data and lessons learned can be used to identify trends, both positive and negative, during the life cycle of the project. Positive trends can indicate areas for organizational process improvement by indicating good practices used on the project that should be adopted throughout the organization. Negative trends and lessons learned indicate areas for needed process improvements throughout the software organization.

4.6.1 Close Software Project or Phase: Inputs

The inputs for closing a project or phase in Section 4.6.1 of the *PMBOK® Guide* are applicable inputs for closing a software project or phase.

4.6.1.1 Project Management Plan

See Section 4.6.1.1 of the *PMBOK® Guide.*

4.6.1.2 Accepted Deliverables

See Section 4.6.1.2 of the *PMBOK® Guide.*

4.6.1.3 Organizational Process Assets

See Section 4.6.1.3 of the *PMBOK® Guide.*

4.6.2 Close Software Project or Phase: Tools and Techniques

The tools and techniques for closing a project or phase in Section 4.6.2 of the *PMBOK® Guide* are applicable to closing a software project product or phase.

4.6.2.1 Expert Judgment

See Section 4.6.2.1 of the *PMBOK® Guide*.

4.6.2.2 Analytical Techniques

See Section 4.6.1.2 of the *PMBOK® Guide*.

4.6.2.3 Meetings

See Section 4.6.2.3 of the *PMBOK® Guide.*

4.6.3 Close Software Project or Phase: Outputs

See Section 4.6.3 of the *PMBOK® Guide* for outputs applicable to closing a software project product or phase.

4.6.3.1 Final Product, Service, or Result Transition

See Section 4.6.3.1 of the *PMBOK® Guide.*

4.6.3.2 Organizational Process Assets Updates

See Section 4.6.3.2 of the *PMBOK® Guide.*

5 SOFTWARE PROJECT SCOPE MANAGEMENT

As stated in the introduction to Section 5 of the *PMBOK® Guide,* Project Scope Management includes the processes required to ensure that the project includes all the work required, and only the work required, to complete the project successfully.

This section of the *Software Extension to the PMBOK® Guide – Fifth Edition* describes the generally accepted principles that are not common to management of most other kinds of projects but are generally accepted principles for managing the scope of software projects. The scope of a software project provides the basis for planning, executing, and measuring and controlling schedule, budget, and resources as well as establishing and maintaining a balance among process attributes (i.e., schedule, budget, personnel, resources) and development of work products (i.e., requirements, design specifications, software code, and verification and validation plans and results).

Project and product scope determine the effort needed to develop or modify a software product; effort is used as the basis to estimate total cost, which includes overhead rates and the cost of additional resources, such as installation, user training, product documentation, and/or an independent testing facility (or verification and validation facility). Effort is used as the basis for determining the schedule for a software project or a project iteration, because software development is accomplished by team effort; a project estimated to require 60 person-months of effort might be scheduled as 10 months for 6 people. See Sections 6 and 7 of this software extension for cost and time management considerations for software projects.

Section 2 of this software extension presents the continuum of life cycles for software projects that lie within a spectrum from predictive life cycles to highly adaptive life cycles. This section of describes the manner in which the scope is managed for various life cycles within the continuum.

Figure 5-1 provides an overview of Software Project Scope Management; it is an adaptation of Figure 5-1 in the *PMBOK® Guide*.

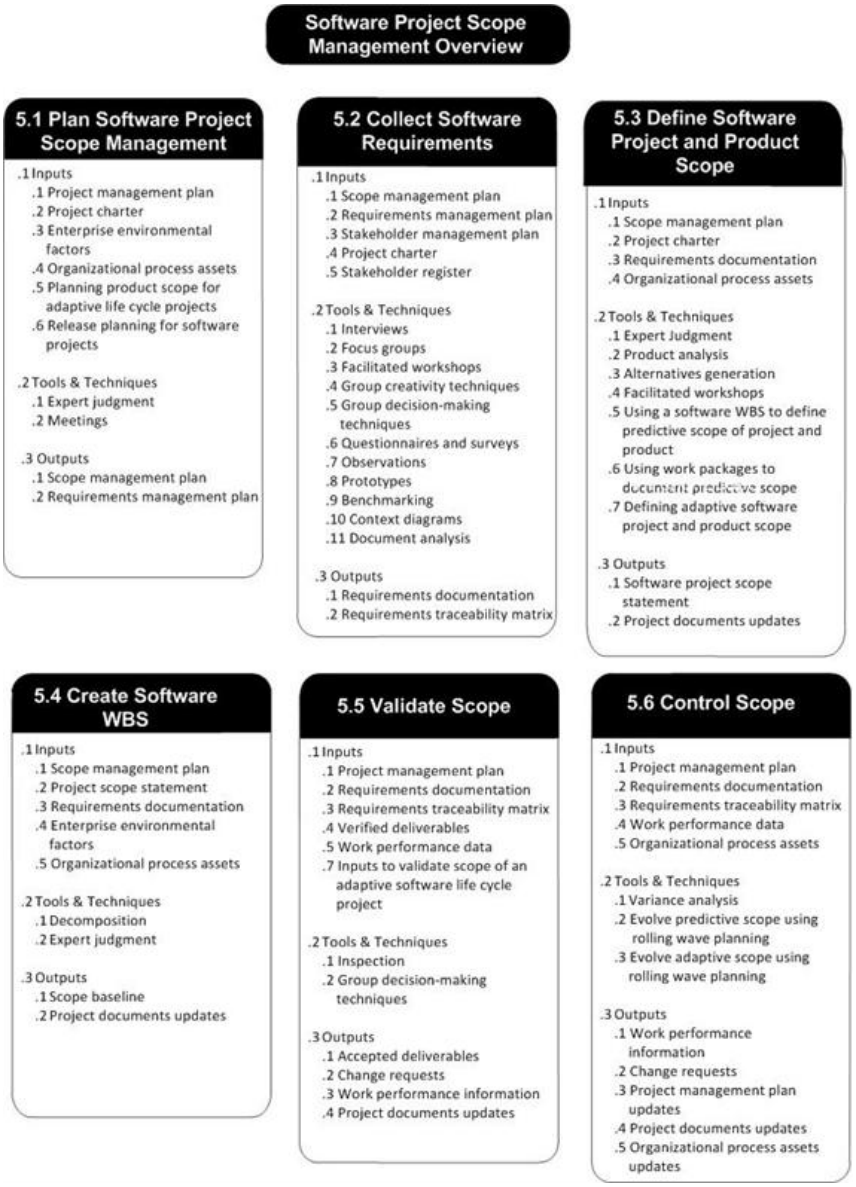


Figure 5-1. Software Project Scope Management Overview

5.1 Plan Software Project Scope Management

Planning scope management for software projects involves determining how the project scope will be defined, validated, and controlled. A scope management plan provides guidance on how scope, for both process and product, will be managed throughout the project. Scope management planning for a software project depends on the life cycle model used to managing the project. Predictive software project life cycles rely on initially collecting and documenting the software product requirements and developing a top-level design; these are used as the basis for developing a work breakdown structure. A predictive life cycle for a software project is most likely to result in a successful project when the product is of a familiar kind and when stable software requirements can be developed in sufficient detail during project initiation and planning. However, many software projects require innovations that cannot be initially foreseen and planned, perhaps because the user

community is not sure what is needed or how it could be provided, or perhaps new technology (new hardware, new infrastructure software) is involved, or perhaps environmental factors such as new policies and regulations are in place. Planning for an adaptive life cycle, in which the requirements and the product scope evolve together, are appropriate for these kinds of software projects.

5.1.1 Plan Scope Management: Inputs

The inputs for planning the scope of a software project life cycle are typically those presented in the *PMBOK® Guide*. In addition to those inputs, two additional inputs are applicable for software projects (see 5.1.1.5 and 5.1.1.6).

5.1.1.1 Project Management Plan

See Section 5.1.1.1 of the *PMBOK® Guide*. A preliminary draft of the project plan is developed and will be refined, based on the project scope.

5.1.1.2 Project Charter

See Section 5.1.1.2 of the PMBOK® Guide.

5.1.1.3 Enterprise Environmental Factors

See Section 5.1.1.3 of the PMBOK® Guide.

5.1.1.4 Organizational Process Assets

See Section 5.1.1.4 of the PMBOK® Guide.

5.1.1.5 Planning Product Scope for Adaptive Life Cycle Projects

The inputs for planning, revising, and expanding the product scope on successive iterations of an adaptive life cycle software project include customer input and the demonstrated results of working software from the previous iteration. Some adjustments to process scope may occur to accommodate the internal organization of the project (as determined by the project team) and the overall scope of the project (as determined by the customer).

5.1.1.6 Release Planning for Software Projects

As illustrated in Figure 5-2, Release planning for adaptive life cycle software projects (those in the iterative-incremental to highly adaptive part of the life cycle continuum) can provide an input for scope management.

As shown, the scope of an overall business goal is specified as a collection of feature sets. Each feature set is decomposed into a set of product increments that will be developed in an iterative manner; each increment will be tested and demonstrated for acceptance. In some cases, it may be possible to specify a release plan during the planning process for a software project. In other cases, the release plan may evolve in a rolling wave manner, as described in Section 5.6.2.3.

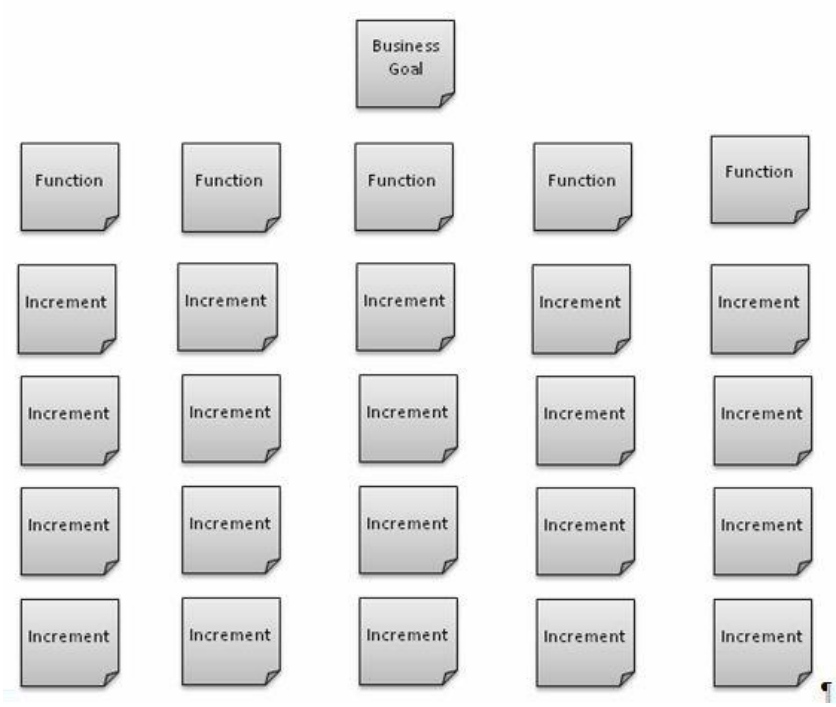


Figure 5-2. Release Planning for an Adaptive Software Project Life Cycle

5.1.2 Plan Scope Management: Tools and Techniques

The tools and techniques used to plan the scope of a software project depend on the nature of the life cycle model used; predictive life cycles and adaptive life cycles use different tools and techniques. Techniques for planning the scope of a predictive life cycle software project include embedding the product structure in the WBS and documenting the work packages needed to accomplish the software development tasks (see Section 5.2). Plans for scope management for adaptive software project life cycles are inherent in the particular adaptive life cycle used.

The tools and techniques in Section 5.1.2 of the *PMBOK® Guide* are equally applicable to planning scope management for a software project.

5.1.2.1 Expert Judgment

See Section 5.1.2.1 of the *PMBOK® Guide.*

5.1.2.2 Meetings

See Section 5.1.2.2 of the PMBOK® Guide.

5.1.3 Plan Scope Management: Outputs

The outputs for planning scope management of a predictive software project life cycle include the scope management plan and the requirements management plan, as indicated in Section 5.1.3 of the *PMBOK® Guide* plus a revised project plan that includes a release plan (see Figure 5-2) or a WBS containing the top-level activities and product components (see Figure 5-3), plus a template for the WBS work packages documented using a template which may take the form illustrated in Figure 5-4 or some other standardized form.

The plan for scope management of an adaptive software project life cycle is implicit in the adaptive life cycle model used, as explained and illustrated in Section 5.3.2.4;

outputs may include a release plan as illustrated in Figure 5-2.

The two outputs for planning scope management in Section 5.1.3 of the *PMBOK® Guide* (5.1.3.1 and 5.1.3.2) are applicable outputs for planning scope management of software projects. They are the scope management plan and the requirements management plan.

5.1.3.1 Scope Management Plan

See 5.1.3.1 of the PMBOK® Guide.

5.1.3.2 Requirements Management Plan

See 5.1.3.2 of the PMBOK® Guide.

5.2 Collect Software Requirements

Software requirements provide the basis for defining and managing the software product scope and also provide a basis for process activities including determination of resources needs, schedule, and budget **[16]**. Predictive life cycles for software projects collect the software requirements during project initiation and planning, to the extent possible; predictive collection of software requirements will be most successful for projects of a familiar kind and when there is experience in working with the customer (who may be either internal or external to the organization). However, many (perhaps most) software projects are characterized by uncertainty and unknown factors during project initiation and planning that results in disruptive, unplanned revision of requirements and work processes throughout the project lifetime. Collection of requirements in an emergent manner as a software project evolves is one of the primary motivations for using an adaptive life cycle model.

5.2.1 Collect Software Requirements: Inputs

For predictive life cycles, the five inputs to collecting software requirements are as stated in Section 5.2.1 of the *PMBOK® Guide*; they include the plans for scope management, requirements management, and stakeholder management plus the project charter and the stakeholder register.

As previously stated, the inputs for collecting software requirements when using an adaptive software project life cycle are inherent in the adaptive model used. The inputs that determine requirements for each iterative cycle include the results of testing and demonstrating the current working version of the software plus the customer's directions, which may take the form of "stories" or features selected from a backlog of users' requirements.

5.2.1.1 Scope Management Plan

See Section 5.2.1.1 of the *PMBOK® Guide*.

5.2.1.2 Requirements Management Plan

See Section 5.2.1.2 of the *PMBOK® Guide*.

5.2.1.3 Stakeholder Management Plan

See Section 5.2.1.3 of the *PMBOK® Guide*.

5.2.1.4 Project Charter

See Section 5.2.1.4 of the *PMBOK® Guide*.

5.2.1.5 Stakeholder Register

See Section 5.2.1.5 of the *PMBOK® Guide*.

5.2.2 Collect Software Requirements: Tools and Techniques

The tools and techniques listed and described in Section 5.2.2 of the *PMBOK® Guide* are equally applicable for both predictive and adaptive software project lifecycles. Prototyping is a particularly effective technique for collecting requirements when using a predictive life cycle. Ongoing demonstration of working software is a primary technique for illustrating progress and collecting the next set of requirements to be implemented when using an adaptive life cycle.

5.2.2.1 Interviews

See Section 5.2.2.1 of the *PMBOK® Guide*.

5.2.2.2 Focus Groups

See Section 5.2.2.2 of the *PMBOK® Guide*.

5.2.2.3 Facilitated Workshops

See Section 5.2.2.3 of the *PMBOK® Guide*.

5.2.2.4 Group Creativity Techniques

See Section 5.2.2.4 of the *PMBOK® Guide*.

5.2.2.5 Group Decision-Making Techniques

See Section 5.2.2.5 of the *PMBOK® Guide*.

5.2.2.6 Questionnaires and Surveys

See Section 5.2.2.6 of the *PMBOK® Guide*.

5.2.2.7 Observations

See Section 5.2.2.7 of the *PMBOK® Guide*.

5.2.2.8 Prototypes

See Section 5.2.2.8 of the *PMBOK® Guide*.

5.2.2.9 Benchmarking

See Section 5.2.2.9 of the *PMBOK® Guide*.

5.2.2.10 Context Diagrams

See Section 5.2.2.10 of the *PMBOK® Guide*.

5.2.2.11 Document Analysis

See Section 5.2.2.11 of the *PMBOK® Guide*.

5.2.3 Collect Software Requirements: Outputs

The outputs in Section 5.2.3 of the *PMBOK® Guide* are applicable for outputs for collecting software requirements. Documentation guidelines for software requirements are presented in (ISO/IEC/IEEE Standard 830 **[17]** and ISO/IEC/IEEE Standard 1362 **[18]**). For adaptive life cycles, the customer is the source of evolving software requirements and the output from collecting software requirements provides the input for the next iterative cycle.

5.2.3.1 Requirements Documentation

See Section 5.2.3.1 of the *PMBOK® Guide*. Requirements traceability is particularly important for software projects because of the lack of physical characteristics of software

5.2.3.2 Requirements Traceability Matrix

See Section 5.2.3.1 of the *PMBOK® Guide*. Traceability provides visibility between software requirements, intermediate work products (e.g., design documentation, test plans, test results) and the deliverable product.

5.3 Define Software Project and Product Scope

According to Section 5.3 of the *PMBOK® Guide*, Define Scope is the process of developing a detailed description of the project and product. The key benefit of this process is that it describes the project boundaries. The inputs, tools and techniques, and outputs for defining the scope of software process and product differ, depending on the position of the selected software project life cycle within the life cycle continuum. The nature of software and the fact that software development is the result of coordinated human effort results in a close integration of process and product scope.

The *PMBOK® Guide* states that since all of the requirements identified in Collect Requirements will not be included in the project, Define Scope involves choosing the requirements that will be part of the product scope. For software projects, this issue is commonly dealt with by prioritizing the requirement using the criteria of value-added and the wants and needs of the customer and user communities. Risks, assumptions, and constraints are also analyzed and added or updated as necessary when defining the software process and product scope.

5.3.1 Define Software Project and Product Scope: Inputs

For predictive software projects, the inputs described in the *PMBOK® Guide* and listed below are used as inputs for defining the detailed product and project scope. For adaptive projects, the project scope is defined by the selected adaptive life cycle; the product

scope evolves during iterative development of the product. The project charter and organizational assets provide inputs for defining software project and product scope for all software project life cycles within the software project life cycle continuum.

5.3.1.1 Scope Management Plan

See Section 5.3.1.1 of the *PMBOK® Guide*.

5.3.1.2 Project Charter

See Section 5.3.1.2 of the *PMBOK® Guide*.

5.3.1.3 Requirements Documentation

See Section 5.3.1.3 of the *PMBOK® Guide*.

5.3.1.4 Organizational Process Assets

See Section 5.3.1.4 of the *PMBOK® Guide*.

5.3.2 Define Software Project and Product Scope: Tools and Techniques

The tools and techniques listed in Section 5.3.2.1 through 5.3.2.4 are applicable for defining the scope of a software project and product. In addition to these, see Sections 5.3.2.5 through 5.3.2.7 for tools and techniques specific to software. For predictive life cycles, a primary tool for defining software project and product scope is the work breakdown structure (WBS). The primary technique is documentation of WBS tasks using work packages. Project and product scope evolve throughout adaptive life cycle projects.

5.3.2.1 Expert Judgment

See Section 5.3.2.1 of the *PMBOK® Guide.*

5.3.2.2 Product Analysis

See Section 5.3.2.2 of the PMBOK® Guide.

5.3.2.3 Alternatives Generation

See Section 5.3.2.3 of the PMBOK® Guide.

5.3.2.4 Facilitated Workshops

See Section 5.3.2.4 of the PMBOK® Guide.

5.3.2.5 Using a Software WBS to Define Predictive Scope of Project and Product

The top level of a WBS depicts the full scope, at a high level, of all the work required, and only the work required, to complete the project successfully, as illustrated in Figure 5-3. The top-level activities are decomposed in a hierarchical manner; the lowest level elements of the WBS are elements of work to be performed (the tasks).

Figure 5-3 does not adequately illustrate the relationships between Manage Project and the other elements of project scope. The organizational chart for the project would show the other elements of project scope as subordinate to Manage Project and would indicate the relationships among the project manager, the software architect/lead designer (technical lead), the customer, and the project team or teams. In some instances, the project manager may be a staff person who provides support for the software architect/lead designer and the project. In those cases, the architect/designer would occupy the superior position in the organizational chart; this is often the case for adaptive software projects.

Some of the software components to be included in a software product may be developed during the software project, some may exist in a library of software components, some may be acquired as open source components, and some may be procured from a software vendor or subcontractor. However, preexisting components typically require modifications to satisfy the product requirements, and thus are seldom used "as is." In all cases, the scope of a predictive software project includes explicit integration, verification, and validation of the initially developed and preexisting components that form the software product, as well as system-level verification and validation of the final product.

The scope of work needed to develop, otherwise obtain, and integrate software components can be embedded in the software project WBS, as depicted in Figure 5-3, which illustrates a partial WBS for developing the software for an automated teller machine; the product components are indicated in bold font; the product scope is thus embedded in the WBS. For purposes of illustration, only the Develop Software element of the WBS is partially decomposed.

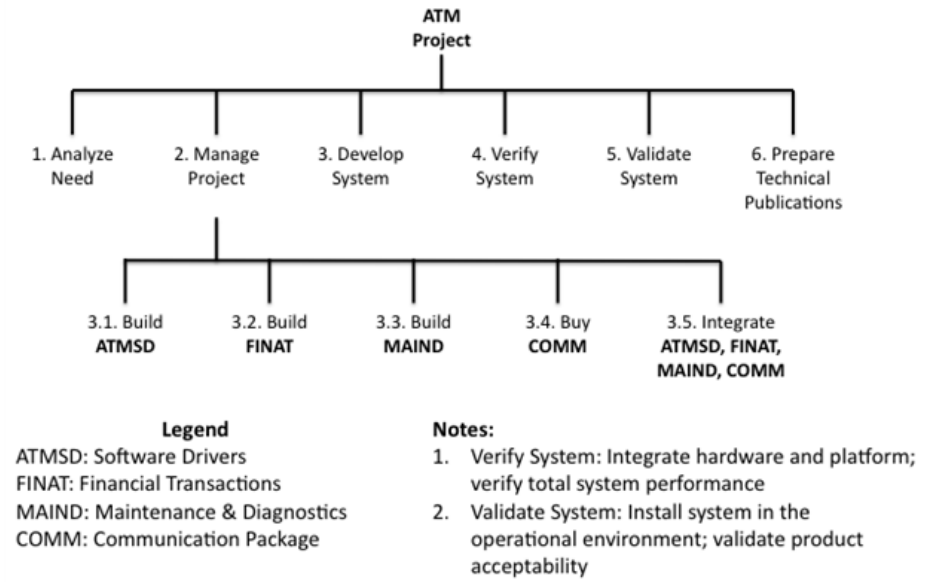


Figure 5-3. Partially Decomposed WBS for an Automated Teller Machine Project

The *PMBOK® Guide* distinguishes between project scope and product scope. For software projects, the two scopes are entwined because of the nature of software and the way in which software is developed or modified by closely coordinated work tasks. The technique of embedding the product structure within a software WBS illustrates the close connection between project scope and product scope for software projects. Adaptive software project life cycles also entwine project scope and product scope, as discussed below.

5.3.2.6 Using Work Packages to Document Predictive Scope

Traditional projects that develop physical artifacts typically use work packages to document the tasks (i.e., the lowest level elements) in the work breakdown structure. Work packages are also used to document the tasks in a software project WBS. The primary factors documented in a software work package are estimated effort (expressed as the product of personnel and duration), estimated duration, software components to be developed or modified, the acceptance criteria for the software components, predecessor and successor tasks, and risk factors that may inhibit successful completion of the software component or components using the allocated effort and additional resources. A template for documenting software WBS work packages is presented in Figure 5-4.

Task ID :	<<WBS number >>
Task identifier:	<<WBS task name>>
Task description:	<<brief description>>
Estimated duration:	<<days or weeks>>
Resources needed:	
Personnel:	<<number of personnel needed to complete this task>>
Skills:	<<personnel skills needed to complete this task>>
Tools:	<<software and hardware needed>>
Travel:	<<to where? for how long?>>
Other:	<<other resources needed to complete this task>>
Predecessor tasks:	<<to be completed before this task can begin>>
Successor tasks:	<<to start after this task is completed>>
Work Products:	<<outputs of this task>>
Baselines:	<<output work products to be placed under version control>>
Risk Factors:	<<potential problems for completing this task>>
Acceptance criteria:	<<for the work products of this task>>

Figure 5-4. Sample WBS Work Package Template for Predictive Software Projects

Because software estimation is an imprecise process, predictive life cycles for software projects typically use two or more techniques to develop estimates. For example, a top-down estimate based on analogies and historical data may be used to determine the scope of overall effort and schedule; these estimated values are allocated to the various elements of the WBS down to the task level. Bottom-up estimation based on expert judgment (perhaps using a Delphi approach) may be used to estimate each task; these estimates are aggregated into a top-level estimate. Variations in the (two or more) estimates are then reconciled. Note that estimates of effort and resource needs can be obtained from a work package using the information in Figure 5-4. A schedule for a collection of work packages can be constructed using the predecessor and successor information in each work package. (See Sections 6 and 7 of this software extension for additional information on time and cost estimation for software projects.)

Effort is the primary cost factor for most software projects because software is the result of intellectual work tasks. For predictive life cycles, effort is typically allocated to work packages in a top-down manner, based on a reconciled estimate of total effort. Effort, being the product of people and time, is decomposed into personnel and time as follows: a specified number of people having specified skills and the estimated time duration needed to complete the work package. For example, a two-person-week task could be specified as one person for two weeks or two people for one week (but not 10 people for one day). Additional resources, such as facilities and software tools, are also specified in each software work package, as needed.

The documented work packages determine the scope of work activities for a predictive software development project. The total project scope for a predictive software project life cycle includes the collective scope of the work packages plus the scopes of the other activities denoted at the top level of the WBS.

5.3.2.7 Defining Adaptive Software Project and Product Scope

The approach to defining project and product scope for an adaptive software project life cycle is implicit in the adaptive life cycle model planned for the project. As described in Section 2.4, the process scope of an iterative-incremental life cycle on the adaptive side of the life cycle continuum includes analysis, design, design partitioning, and constructing, testing, and demonstrating progressive increments of growing product capabilities on each iterative cycle of software construction, perhaps using rolling wave elaboration of the WBS to progressively define the product scope as the project evolves and as the design and design partitioning are revised (see Section 5.6.2.2 for a discussion of rolling wave elaboration of a software WBS).

In adaptive life cycle software projects, the scope of the product is determined as a succession of outcomes; small increments ("stories") deliver features that build toward a business goal or product vision in a cascading manner. The outcomes may be based on a release plan, as illustrated in Figure 5-2 or they may emerge as the project evolves. As the project evolves, the details of product scope are clarified while maintaining integrity with the desired outcome.

The translation of Goals to Features to Product Increments is illustrated in Figure 5-5, where incremental implementation of a succession of features is implemented as a sequence of small increments.

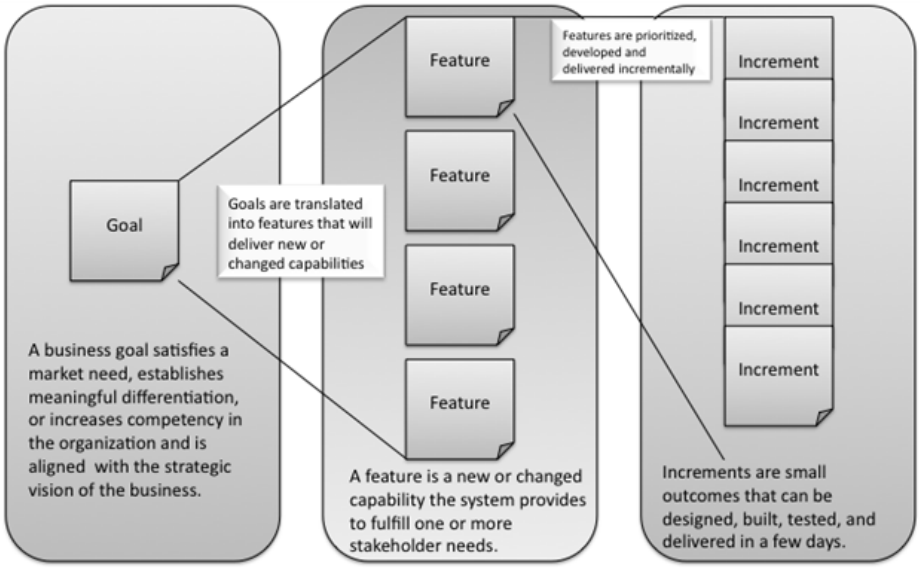


Figure 5-5. Delivery of Functionality as a Succession of Product Increments

For an adaptive project life cycle model, the process scope expands to accommodate frequent iterations of product construction, testing and refactoring, and demonstration, as illustrated in Figure 5-6 for a highly adaptive software project life cycle project (see also Figure 2-7 and the related discussion). The product scope expands during the iterations, at the direction of the customer.

The scope of requirements that can be implemented during an adaptive development cycle is based on the story or stories to be implemented, the specified time period, and the productivity of the development team. The estimated productivity is based on accumulated experience using measures such as velocity and burn down rate. Short-duration development cycles provide rapid feedback and the ability to easily revise and adjust the product scope based on demonstrations of working software.

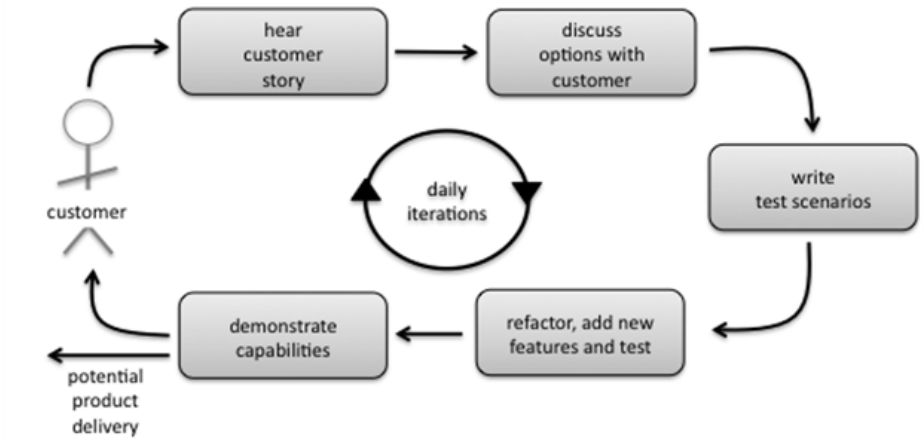


Figure 5-6. A Highly Adaptive Life Cycle for a Software Project

Additional aspects of the adaptive life cycle illustrated in Figure 5-6 are the development of requirements-based test scenarios before writing the software code and refactoring3 of the software structure, which may be necessary to better accommodate the code for the additional features being added. Also note in Figure 5-6 the option of delivering working software into the user environment, at the end of each development cycle, which is possible because each software version, after initial start-up, results in working, deliverable software.

Another aspect common to all adaptive life cycles is emphasis on a learning environment in which customers and users clarify and prioritize requirements in an emergent manner, based on value-adding priorities and periodic demonstrations of working software.

5.3.3 Define Software Process and Product Scope: Outputs

The outputs for defining scope in Section 5.3.3 of the *PMBOK® Guide* are equally applicable for defining software process and product scope. The modifications to 5.3.3.1 also applies.

5.3.3.1 Software Project Scope Statement

For both predictive and adaptive software project life cycles, the statement of software project scope describes deliverables and the work required to produce them. The project scope statements for both predictive and adaptive software project life cycle projects

typically include a statement of assumptions, constraints on the project plus exclusions for the project, and product attributes that are outside the project's boundaries.

Product acceptance criteria should also be included in the project scope statement. For predictive life cycles, the acceptance criteria are provided in each work package that produces a work product (see Figure 5-4). For adaptive life cycles, the acceptance criteria for the next product increment are stated at the beginning of the iterative cycle that will produce the increment. In addition, the process scope may evolve for adaptive life cycle projects. In an ideal predictive life cycle the scope statement would be a static document, although this is rarely the case. The project scope statement is planned to be an evolving document for adaptive life cycle software projects. Planning for the inevitable evolution of project scope is a primary factor that distinguishes adaptive software project life cycles from predictive life cycles.

5.3.3.2 Project Documents Updates

See Section 5.3.3.2 of the PMBOK® Guide.

5.4 Create Software WBS

The inputs, tools and techniques, and outputs for Create WBS in the *PMBOK® Guide* are equally applicable to creating work breakdown structures for predictive software project life cycles. Some considerations for developing a WBS for a predictive software project include:

The WBS can be developed top-down, by first specifying the project scope at the top level and decomposing each top element into details; or by first identifying low-level tasks to be performed and grouping them into successively larger groupings (activities); or by working "middle out" by identifying intermediate-level activities and decomposing them downward and grouping them upward. In practice, all three approaches are typically used to produce the WBS. Predefined templates for work breakdown structures and work packages, tailored to fit local needs, make the task of constructing a software WBS much easier than starting "from scratch."

The *PMBOK® Guide* distinguishes between organizing a WBS by project phase or by product deliverables. Using the technique of embedding the product structure in the WBS, as described in Sections 5.3.2.5 and 5.6.2.2 removes this distinction for software projects.

Section 5.4.3.1 of the PMBOK® Guide indicates that a WBS dictionary should be developed as part of the scope baseline. The work package template provided in Figure 5-4 of this software extension contains most of the information needed as inputs for a WBS data dictionary, as specified in the *PMBOK® Guide.*

5.4.1 Create WBS: Inputs

The inputs in Section 5.4.1 of the *PMBOK® Guide* for creating a WBS are equally applicable for creating a software WVBS.

5.4.1.1 Scope Management Plan

See Section 5.4.1.1 of the *PMBOK® Guide*.

5.4.1.2 Project Scope Statement

See Section 5.4.1.2 of the *PMBOK® Guide*.

5.4.1.3 Requirements Documentation

See Section 5.4.1.3 of the *PMBOK® Guide*.

5.4.1.4 Enterprise Environmental Factors

See Section 5.4.1.4 of the *PMBOK® Guide*.

5.4.1.5 Organizational Process Assets

See Section 5.4.1.5 of the *PMBOK® Guide*.

5.4.2 Create WBS: Tools and Techniques

The tools and techniques for creating a WBS described in the *PMBOK® Guide* are equally applicable for creating a software WVBS

5.4.2.1 Decomposition

See Section 5.4.2.1 of the *PMBOK® Guide*. The decomposition technique for creating a WBS, as described is equally applicable for creating a software WVBS.

5.4.2.2 Expert Judgment

See Section 5.4.2.2 of the PMBOK® Guide.

5.4.3 Create WBS: Outputs

The outputs for creating a WBS in Section 5.4.3 of the *PMBOK® Guide* are equally applicable for creating a software WVBS. In addition the software WBS is an output from creating a software WVBS.

5.4.3.1 Scope Baseline

See Section 5.4.3.1 of the PMBOK® Guide.

5.4.3.2 Project Documents Updates

See Section 5.4.3.2 of the PMBOK® Guide.

5.5 Validate Scope

Validate Scope covers formalizing acceptance of the completed project deliverables. In software engineering, a distinction is made between verification and validation. Verification is concerned with determining, in an objective manner, that the deliverable software is correct, complete, and consistent with respect to the product requirements, design constraints, and other product parameters. Validation is concerned with determining, in an objective manner, that the delivered software satisfies the needs and expectations of customers, users, and other stakeholders when installed in the operational environment. Colloquially, verification answers the question “did we build the software correctly?” and validation answers the question, “did we build the correct software?”

5.5.1 Validate Scope: Inputs

Inputs to validating the scope of software deliverables includes all of the deliverables, as specified in the scope plan. The primary deliverable is working software but other deliverables may include user training materials and usage information, installation and operating instructions, and guidance for maintainers. The intended users, operations, and maintainers are those who validate these inputs for acceptability. These work products may be in textual hard copy or accessible online. In some cases a suite of development work products including some or all of requirements, design documentation, traceability matrices, and test plans and test results may be inputs for validating scope.

For predictive software project life cycles, inputs for validating the delivered software include the requirements, one or more requirements traceability matrices, design documentation, the software code, and the software validation plan. For software, a distinction is made between the external stakeholders' requirements, which are typically documented in a Concept of Operations (ConOps)4 and the technical specifications, which are the translation (to the extent possible) of external requirements into objectively quantified parameters that must be satisfied in order to satisfy the external requirements. The ConOps may also be known as an Operational Concepts Document (OCD), a marketing analysis, or some equivalent name.

ISO/IEC/IEEE Standards 830] and 1362] provide guidance for preparing the ConOps and the technical specifications. A traceability matrix is typically used to trace between the external stakeholders' requirements and the technical specifications and one or more traceability matrices is used to establish the correspondences between technical specifications and design documentation, software code, details of the validation plan, and validation results.

The five inputs for validating scope in Section 5.5.1.1 to 5.5.1.6 of the *PMBOK® Guide* are applicable for validating the scope of predictive software project life cycle projects. They are the scope management plan, scope baseline, requirements documentation, requirements traceability matrix, validated deliverables, and work performance data.

The additional input described in 5.5.1.6 is concerned with inputs for validating the scope of adaptive software project life cycle projects.

5.5.1.1 Project Management Plan

See Section 5.5.1.1 of the *PMBOK® Guide*.

5.5.1.2 Requirements Documentation

See Section 5.5.1.1 of the *PMBOK® Guide*.

5.5.1.3 Requirements Traceability Matrix

See Section 5.5.1.1 of the *PMBOK® Guide*.

5.5.1.4 Verified Deliverables

See Section 5.5.1.1 of the *PMBOK® Guide*.

5.5.1.5 Work Performance Data

See in Section 5.5.1.1 of the *PMBOK® Guide*.

5.5.1.6 Inputs to Validate Scope of an Adaptive Software project life cycle Project

For adaptive software project life cycle projects, validation occurs incrementally during and at the end of each iterative cycle; the inputs are the test cases and demonstration scenarios developed before and during each iterative cycle. Inputs for adaptive life cycles may include formally documented requirements, one or more requirements traceability matrices, design documentation, and the software code, all of which are updated incrementally as they evolve during iterative cycles. For example, user stories and the associated technical specifications are typically recorded and tracked using a version control tool that captures them, iteration by iteration. A formal validation plan may be developed initially and applied throughout the project life cycle or validation may be an element that is built into each iterative cycle without a formal validation plan.

For adaptive software lifecycles, some validation tools and techniques, such as formal inspections and demonstrations, are applied by, and the results observed by the software developers. End-of-cycle demonstrations of working software are conducted for the designated customer and appropriate external stakeholders.

5.5.2 Validate Scope: Tools and Techniques

The tools and techniques for validating scope in Section 5.5.2 of the *PMBOK® Guide* (5.5.2.1 and 5.5.2.2) are applicable for validating the scope of both predictive and adaptive software project life cycle projects. Testing and demonstrations are also useful tools and techniques for validating the scope of a software product. A software inspection is a formalized review process that involves preparation for the inspection; the roles to be played by the inspectors; checklists and forms; a moderated meeting; and documented follow up activities. Other forms of reviews include peer reviews of working software, management reviews of validation status, and reviews with external stakeholders.

Ideally, a software test involves preparation of test inputs, test conditions, and an objective statement of specified test results; execution of the test in a specified environment under specified conditions; and observation and recording of the test results.

A validation demonstration differs from a validation test, in that a test has objectively stated success criteria whereas a demonstration relies on the subjective observations of witnesses to determine the success or failure of the demonstrated software features. As stated above, some demonstrations are conducted for developers and some are for the designated customer and appropriate external stakeholders.

For predictive life cycle software projects, validation is a major phase of the software project that occurs at the end of the software development or software modification project. For an iterative-incremental life cycle project, validation occurs at the end of each iteration and at the end of the project. For adaptive life cycles, validation occurs during and at the end of each iterative cycle. A major validation effort may accompany delivery of the final product, at the end of the final iterative cycle.

5.5.2.1 Inspection

See Section 5.5.2.1 of the *PMBOK® Guide*.

5.5.2.2 Group Decision-Making Techniques

See Section 5.5.2.2 of the *PMBOK® Guide*.

5.5.3 Validate Scope: Outputs

The outputs for Validate Scope from the *PMBOK® Guide* are applicable to software.

Change requests may be handled informally or may be handled by a process for review and disposition using a Perform Integrated Change Control process (Section 4.5 of the *PMBOK® Guide*), depending on the formality of the validation process.

5.5.3.1 Accepted Deliverables

See Section 5.5.3.1 of the *PMBOK® Guide*.

5.5.3.2 Change Requests

See Section 5.5.3.2 of the *PMBOK® Guide*. Change requests may be handled informally or may be handled by a process for review and disposition using a Perform Integrated Change Control process (Section 4.5 of the *PMBOK® Guide*), depending on the formality of the validation process.

5.5.3.3 Work Performance Information

See Section 5.5.3.3 of the *PMBOK® Guide*.

5.5.3.4 Project Documents Updates

See Section 5.5.3.4 of the *PMBOK® Guide*.

5.6 Control Scope

Control Scope is the process of monitoring the status of project and product scope and managing changes to the scope baseline. The inputs, tools and techniques, and outputs used to control the scope of a software project vary with the life cycles along the continuum of life cycle models.

5.6.1 Control Scope: Inputs

The inputs for controlling scope in Section 5.6.1 of *PMBOK® Guide*), are applicable to controlling the scope of a predictive life cycle software project. A software project based on an iterative-incremental life cycle relies on demonstrations of increasing product capability at the end of the iterations as the primary input for assessing and controlling scope. Iterative-incremental projects on the predictive side of the life cycle continuum use more formal mechanisms to obtain inputs for scope control than do iterative-incremental projects on the adaptive side of the continuum.

Adaptive life cycle projects typically use short iteration cycles and frequent demonstrations of progress to provide the input for ongoing control of process and product scope. The customer, in consultation with the software development team, determines the requirements and the time period for developing each successive version of the product. The period of time boxing for iterations typically varies from one day to a few weeks, depending on the customer’s wishes and the particular adaptive life cycle being used.

5.6.1.1 Project Management Plan

See Section 5.6.1.1 of the *PMBOK® Guide*.

5.6.1.2 Requirements Documentation

See Section 5.6.1.2 of the *PMBOK® Guide*

5.6.1.3 Requirements Traceability Matrix

See Section 5.6.1.3 of the *PMBOK® Guide.*

5.6.1.4 Work Performance Data

See Section 5.6.1.4 of the *PMBOK® Guide.*

5.6.1.5 Organizational Process Assets

See Section 5.6.1.5 of the *PMBOK® Guide.*

5.6.2 Control Scope: Tools and Techniques

Predictive life cycles for software projects typically use traditional project management technique for controlling scope, including milestone reviews, change requests, and change control boards, in addition to variance analysis, as described in Section 5.6.2 of the *PMBOK® Guide.* Additional tools and techniques for controlling the scope of a software project are provided in Sections 5.6.2.2 and 5.6.2.3.

As previously stated, a predictive, plan-driven life cycle for a software project is most likely to result in a successful project when the product is of a familiar kind, when stable software requirements can be developed in sufficient detail during project initiation and planning, and when the customer is a familiar one. However, many software projects require innovations that cannot be initially foreseen and planned, perhaps because the user community is not sure what is needed or how it could be provided, because the customer is new, or perhaps because new technology (new hardware, new infrastructure software) is involved.

An important aspect of adaptive life cycles for software projects is that the customer, in consultation with the development team, determines the scope of product elements to be included in each development cycle. The requirements are scoped to accommodate the time and resources available to construct the software for those requirements. The scope of the product continues to expand during each development cycle until the customer requirements are fully satisfied, or until time and resources are exhausted. In the latter case, the working, deliverable software will incorporate the most value-adding user features, which were specified by the customer as input to the iterative cycles.

The overall project scope, including schedule and budget for an adaptive life cycle project, may be fixed or may grow or shrink adaptively based on value-added considerations of continuing or terminating product development. To reiterate, an adaptive software project can deliver the software at the end of any iterative cycle after initial startup because working deliverable software at the end of each iteration is a fundamental aspect of adaptive software project life cycles.

It should also be noted that the scope of an adaptive life cycle includes other elements of project scope as appropriate to the needs of the project, such as a scope management plan, initial architectural design, independent verification and validation, configuration management, and quality assurance and quality control. The continuum of software project life cycles is not a thin line, but is multidimensional to accommodate additional aspects of scope control.

5.6.2.1 Variance Analysis

See Section 5.6.2.1 of the PMBOK® Guide.

5.6.2.2 Evolve Predictive Scope Using Rolling Wave Planning

Every software project results in a unique product, either new or modified, because replication of existing software is a trivial process, as compared to replication of physical artifacts. Most software projects thus require innovation and creative problem solving to satisfy new and evolving needs. The scope of the project and the product thus evolve during the project lifetime. For predictive software projects, the details of scope evolve (within the constraints of overall scope) by evolving the WBS in a rolling wave manner; increased understanding of the problem to be solved permits elaboration of the WBS. Some rolling wave modifications of a software project WBS may be accomplished within the overall scope constraints of schedule, budget, resources, and technology, while other modifications may require renegotiation of the scope constraints.

Rolling wave elaboration of a software WBS is typically accomplished periodically, perhaps monthly, to accommodate increased understanding of the problem to be solved. Rolling-wave elaboration also may be accomplished as circumstances dictate, such as changes to requirements, schedule, budget, resources, or technology.

An example of rolling wave elaboration of the WBS for the ATM project depicted in Figure 5-3 is illustrated in Figure 5-7, where the details of building the financial transaction component have been added to Figure 5-3, perhaps after some prototyping and feasibility analysis. The work package for the Financial Transaction component in Figure 5-3 is decomposed in Figure 5-7 into work packages for the four subordinate software components plus the FINAT integration task. Note, as before, that the product components are denoted in boldface font. Also, note the decision to reuse existing recorder software from another software product. Software component development and integration may progress in an incremental manner before all of the components are completed. This approach would likely be used for iterative-incremental life cycles on the predictive side of the predictive to adaptive continuum (see Section 2.4 of this extension).

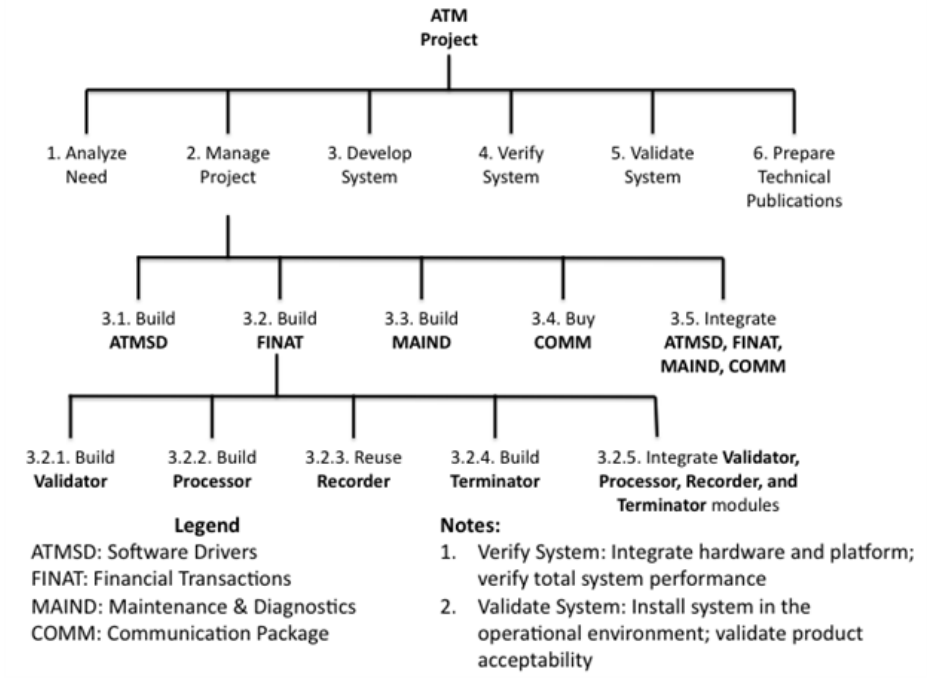


Figure 5-7 . Rolling Wave Elaboration of the ATM WBS in Figure 5-3.

In summary, predictive software project life cycles for software projects attempt to initially develop software requirements that are complete, correct, consistent, and detailed. The requirements provide the basis for determining the scope of the project and for developing the WBS and the work packages. Project scope is then managed by carefully controlling changes to the software requirements and the work activities need to implement those requirements. The impact of changes to product requirements on the project schedule, budget, resources, and technology may require revision of the project scope and is reflected in changes to the WBS. Change control boards and a version control system are typically utilized to manage the changing scope of a software project when using a predictive software project life cycle.

5.6.2.3 Evolve Adaptive Scope Using Rolling Wave Planning

The scope of adaptive life cycle software projects can also evolve in a rolling wave manner, as illustrated in Figure 5-8, which is the adaptive equivalent of a rolling wave WBS. The “boxes” in each quarter (Q1 – Q4) are increments of functionality, as shown in Figure 5-2.

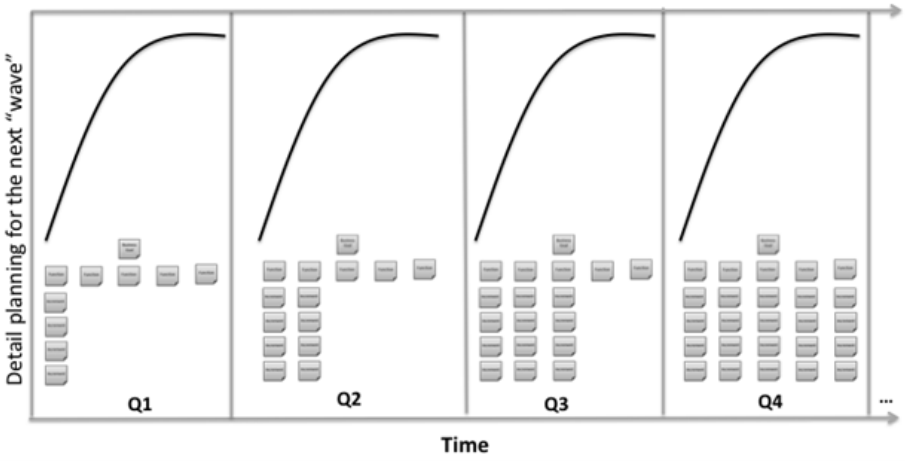


Figure 5-8. Rolling Wave Elaboration for an Adaptive Software project life cycle Project

As shown, a business goal may be partially realized by a function (a set of features) that is decomposed into increments during Q1. The “rising curve” indicates planning for Q2 during Q1. The next feature set (function) is instantiated in Q2 and while the Q3 features are being planned. The cycle continues until the business goal is satisfied.

5.6.3 Control Scope: Outputs

The outputs for controlling scope in Section 5.6.3 of the *PMBOK® Guide* are applicable to controlling the scope of a software project.

For software projects, the outputs of scope control vary with the life cycle used along the continuum of software project life cycle models. For predictive life cycles, the primary outputs of scope control are the decisions of the change control board to deny or accept change requests (the project manager and the software architect are members of the CCB); acceptance may be scheduled for immediate or delayed response. For adaptive life cycles, the primary output of scope control is the decision of the customer concerning the next set of requirements to be implemented or the changes to be made to the currently working software. The development team may, in consultation with the customer, spend the next interim period modifying the software architecture and doing significant refactoring of the existing software base before continuing the iterative software development cycles.

Independent of the life cycle used within the continuum of life cycles, the output of scope control may require the project manager, higher management, and the customer (or customers) to make significant changes to process parameters (schedule, budget, resources) and product parameters (requirements, technology, mission). These changes may be required by factors that are beyond the control of the project manager, such as a changing operational environment, changes in the organization's strategic vision, changes in technology or infrastructure, or changes to competitors' products.

5.6.3.1 Work Performance Information

See Section 5.6.3.1 of the *PMBOK® Guide*.

5.6.3.2 Change Requests

See Section 5.6.3.2 of the *PMBOK® Guide*.

5.6.3.3 Project Management Plan Updates

See Section 5.6.3.3 of the *PMBOK® Guide*.

5.6.3.4 Project Documents Updates

See Section 5.6.3.4 of the *PMBOK® Guide*.

5.6.3.5 Organizational Process Assets Updates

See Section 5.6.3.5 of the *PMBOK® Guide*.

6 Software Project Time Management

Section 6 of the *PMBOK® Guide* includes seven processes that constitute Project Time Management, which as stated in the introduction to Section 6 include the processes required to manage the timely completion of the project. This section of the *Software Extension to the PMBOK® Guide – Fifth Edition* indicates the activities in the *PMBOK® Guide* that are applicable to time management for software projects and describes approaches that are especially important for time management of software projects.

Project Time Management for software projects is driven by risk, resource availability, business value, and the scheduling method(s) used. Software project schedules should remain flexible throughout the life cycle of a software project to adjust to evolving

stakeholder understanding of value and risk. Understanding the different scheduling methods and selecting one or more that are appropriate for dealing with scheduling risks of a software project are critical for success. Because most of the development cost for a software project is human resource effort, and because effort is the product of people and time, this section on software project time management is closely linked to Section 7 of this software extension.

Key time management decisions depend on the schedule management plan that selects a scheduling methodology, a scheduling tool, and sets the format and establishes criteria for developing and controlling the project schedule. Scheduling methods, effort estimation, and time management procedures that are appropriate for software projects are discussed in this section. All of these require life cycle (see Section 2.4) and scope (see Section 5) decisions that support their implementation.

Primary factors in selecting a scheduling method are risk, project environment, product architecture, and the different value propositions associated with each software capability to be implemented.

Selecting a scheduling method, like most software decisions, should consider the risks associated with the project and the environment, including organization culture and organizational process assets. Using risk to determine the scheduling method is dependent on the risk-related processes addressed in Section 11. For example, one significant risk for a project is the inability to provide immediate value, if initially reduced, to the customer to gain trust. In that situation, a scheduling method dependent on a predictive life cycle (big design up front) would be very risky. On the other hand, if human life or the company's reputation is at stake and if the product is not as good as it could be or is in the process of being certified, more design upfront and increased priority on quality-related processes (see Section 8) is warranted.

The project environment is a significant influence on the appropriateness of a scheduling method. If the method is not supported by the culture of the organization or is not aligned with the management infrastructure and incentives, the project may fail to meet schedule commitments regardless of any risk-mitigating effects it might provide.

Software project scheduling is usually dependent on the existence and maturity of product architecture. The constraints and design attributes that are established by architecture may need to be developed first, and therefore it may not be difficult and risky to downstream work until the architecture is mature. Architecture is also an important enabler for early value delivery. Product architecture is a critical contributor to the ability to progressively deliver value and to allow the project team to respond to change late in a project, especially in the case of adaptive life cycles (iterative-increment to highly adaptive).

Value for the customer is an important factor in all projects. However, in most cases, the nature of software allows value to be provided in installments rather than in a single lump sum; this enables the scheduling method to take advantage of time-related or changing values. It may be that highly valuable capabilities can be provided early in a project rather than only at completion. It may also be possible to provide less valuable capabilities while waiting for more valuable capabilities to mature. However, even if the customer is able to use partial functionality, this option is eliminated if the scheduling method and the architecture are not structured to take advantage of progressive installments of value.

In addition to the methods described in Section 6 of the *PMBOK® Guide*, examples of other scheduling methods used for software projects include:

- 0**Structured scheduling.** This method involves a rigorous, structured approach and encompasses traditional scheduling methods described in the PMBOK, CMMI, and other project governance models. This type of scheduling can be used when some or most of the following conditions apply: well-understood product requirements; related precedent work within the

organization; strict architectural requirements; specific limits on the number, size, or frequency of deployments; critical backward compatibility issues; or heavy dependencies on new infrastructure. This rigor is often needed for products with high safety, security, or regulatory constraints; for major version releases of high-profile products (e.g., MS Office, SAP or Oracle), or for very large projects with significant amounts of multi-group coordination.

- 0**Schedule as independent variable (SAIV).** This is a date-certain scheduling method. It is used when there is a specific date after which the value of the project declines precipitously. Examples are time to market considerations, an immovable event for which the product is required (trade show, scheduled version release), or a date by which the enterprise is required to institute some element of a regulatory change. SAIV is characterized by careful prioritization of functional requirements and the strategy of ensuring availability of the most valued functionality by the required deadline. This is the same concept as "time boxing," where time constrains the work done within an iteration or incremental completion. More discussion is provided in Section 6.3.2.5.

- 0**Iterative planning with a backlog.** This is a form of rolling wave planning used in adaptive projects, where the requirements are prioritized, roughly allocated into iterations, and iterations are refined just prior to execution. Iterative planning continues throughout the project life cycle and is central to the ongoing learning and adaptation that occurs on emergent projects. This approach is often used to deliver value more quickly to the customer or when there are a large number of functions that need to be implemented, however, there are few dependencies between functions. This fits many types of projects, as shown by the popularity of adaptive life cycle models.

- 0**On-demand scheduling.** Based on the theory-of-constraints and lean manufacturing pull-based scheduling concepts, this approach does not pre-schedule a project using increments, but directly pulls work from a backlog or intermediate queue of work immediately as resources become available. It provides the customer with a statistically based estimate of time-to-complete for any given task (often referred to as lead time), and can be used to continually assess backlogged task-value to maximize value for customers. On-demand scheduling is often used for systems that are evolved incrementally (operational or sustainment environments), where tasks may be made relatively similar in size and scope, or can be bundled in such manner. It may use classes of service to allow resource utilization flexibility, while ensuring that critical or time-sensitive tasks are "fast-tracked," and necessary but not schedule-critical tasks are not put off indefinitely. The goals of such approaches are for work to take the least amount of time, estimation activities are minimized, and the value of the work accomplished maximized.

- 0**Portfolio management scheduling.** This method follows the hierarchical practices of defining the software investment as a priority of work for a project team where that sequence is valued by parameters set by the organization. Scheduling is not based on the size of the activity, but rather on its importance to the organization. Estimation using this method is based upon the value created vs. the time and/or cost of the development project. This method is often used for strategic management of large enterprise systems or commercial services. See SWX Section 4.1 for more information. These methods form the foundation for much of the remainder of this section.

Figure 6-1 provides an overview of Software Project Time Management; it is an adaptation of Figure 6-1 in the *PMBOK® Guide*.

Figure 6-1. Software Project Time Management Overview

6.1 Plan Software Project Schedule Management

The inputs, tools and techniques, and outputs presented in Section 6.1 of the *PMBOK® Guide*

are generally applicable to planning a software project schedule, when the issues outlined previously are taken into account. Enterprise environmental factors (Section 6.1.1.3) that may impact planning software project schedule management include software project portfolios and enterprise architectures. Organizational process assets (Section 6.1.1.4) may include governance documents and project life cycles predefined for use within the software development organization.

6.1.1 Plan Software Project Schedule Management: Inputs

The inputs in Section 6.1.1 of the *PMBOK® Guide*) are applicable to for planning software project schedule management.

6.1.1.1 Project Management Plan

See Section 6.1.1.1 of the *PMBOK® Guide*.

6.1.1.2 Project Charter

See Section 6.1.1.2 of the *PMBOK® Guide*.

6.1.1.3 Enterprise Environmental Factors

See Section 6.1.1.3 of the PMBOK® Guide.

6.1.1.4 Organizational Process Assets

See Section 6.1.1.4 of the *PMBOK® Guide*

6.1.2 Plan Software Project Schedule Management: Tools and Techniques

The tools and techniques in Section 6.1.2 of the *PMBOK® Guide* are applicable for planning software project schedule management.

6.1.2.1 Expert Judgment

See Section 6.1.2.1 of the *PMBOK® Guide*.

6.1.2.2 Analytical Techniques

See Section 6.1.2.2 of the *PMBOK® Guide*.

6.1.2.3 Meetings

See Section 6.1.2.3 of the PMBOK® Guide.

6.1.3 Plan Software Project Schedule Management: Outputs

The output in Section 6.1.3 of the *PMBOK® Guide* is applicable for planning software project schedule management.

6.1.3.1 Schedule Management Plan

See Section 6.1.3.1 of the PMBOK® Guide.

6.2 Define Software Project Activities

Defining activities in software projects is based on the functional and nonfunctional requirements, project scope, and the project environment. These are also determined by the development approach and project life cycle selected.

As described in Section 2, software project activities and processes are detailed in ISO/IEC 12207:2008. Many of these processes and activities are included in the *PMBOK® Guide* and are also addressed in this software extension. Software activities are not rigidly ordered and may be performed sequentially, iteratively, concurrently, or in whatever way best meets the needs and risks of the project. Section 2 provides additional information on project life cycles and how they are selected for a project.

6.2.1 Define Software Project Activities: Inputs

The inputs for defining project activities in Section 6.2.1 of the *PMBOK® Guide* are generally applicable to defining inputs for software project activities. In addition to these, Section 6.2.1.5 is applicable to defining software project activities. Other factors that may influence the definition of software project activities include existing work orders and enhancement requests; technical debt remaining from previous iterations, incomplete functionality and needed rework; and external activities such as database or operating system upgrades and business process changes.

6.2.1.1 Schedule Management Plan

See Section 6.2.1.1 of the *PMBOK® Guide*.

6.2.1.2 Scope Baseline

As stated previously, an organization's enterprise architecture, if there is one, is an environmental factor that may influence the definition of software project activities.

6.2.1.3 Enterprise Environmental Factors

See Section 6.2.1.3 of the *PMBOK® Guide*.

6.2.1.4 Organizational Process Assets

In addition to those in Section 6.2.1.4 of the *PMBOK® Guide,* organizational process assets that may influence defining project activities include governance documents, project life cycle models, team velocity measures and other productivity measures such as SAIV described in the introduction to Section 6 of this extension, and cadence or flow measures such as time-in-process statistics for adaptive and on-demand scheduling.

6.2.1.5 Organization Parameters

Organization parameters provide metadata requirements for software projects so that projects, programs, and portfolio information can be rolled up into scorecards and strategic plans. Understanding and using these parameters will ease project reporting and eliminate rework by the portfolio manager.

6.2.2 Define Software Project Activities: Tools and Techniques

The tools and techniques for defining project activities in Section 6.2.2 of the *PMBOK® Guide* are generally applicable for defining the tools and techniques for software project activities. In addition to these, Sections 6.2.2.5 through 6.2.2.7 apply to defining software project activities.

6.2.2.1 Decomposition

See Section 6.2.2.1 of the *PMBOK® Guide*.

6.2.2.2 Rolling Wave Planning

See Section 6.2.2.2 of the *PMBOK® Guide*.

6.2.2.3 Expert Judgment

See Section 6.2.2.3 of the *PMBOK® Guide*.

6.2.2.4 Feature or Story Breakdown Structures

As with most other kinds of projects, work activities for software projects may be assigned to different life cycle phases or executed by different elements of a project team. For example, activities could be defined and assigned to dedicated resources within the traditional phases of requirements, design, implementation, validation, and transition, as in a predictive life cycle. In contrast, a software project manager may group work activities into clusters associated with some required capability. Given an enabling architecture, a capability may be developed as a single activity integrating most or all of the traditional, predictive life cycle activities. These capabilities may be called features or stories and represent some part of the software product's functional and nonfunctional requirements as one entity.

Complex stories may be defined as *epics*, described at a high level, and refined into stories at a later date. Stories that are associated by a common factor, such as functionality, data source, security level, or use case, may be associated within a *theme*. All of these are ways to define work activities that will deliver software capabilities. Other project work (procurement, documentation, risk management, training, etc.) may also be referenced as stories or features and tracked and managed using backlogs.

Defining software project activities in this manner provides a number of benefits, as follows:

- 0Holistic estimation and value assessment of capabilities in terms of user desires,
- 0Progressive delivery of value through incremental capability deployment,
- 0More rapid learning and progressive elaboration of user needs,
- 0Quicker response in changing environments,
- 0Reduced late rework when it is most expensive, and
- 0Within a given feature or capability development, lessons learned and the resulting process improvement actions can be quickly identified and adopted, thereby immediately impacting project productivity.

More information on this can be found in the discussion in Section 2 of this extension.

6.2.2.6 Storyboards

Storyboards for defining software capabilities are similar to those used in television and

movie production. They provide a graphical overview of the project that illustrates the order in which certain capabilities or user stories are to be completed.

6.2.2.7 Use Cases

Use cases describe a software product's functionality from the user's point of view. Use cases provide scenarios of operation (step-by-step interactions) between a user and the product for a desired functional capability. A use-case scenario can be specified using an itemized list of steps or by using a UML/SysML notation such as a sequence diagram, activity diagram, or state diagram. Software tools are available for these notations; the tools support various forms of analysis and elaboration of the diagrams for software implementation. Use cases that have alternate paths, exceptions, and business rules, in addition to the primary scenario, can be used to slice use case stories into features to be implemented, as shown in Figure 6-2.

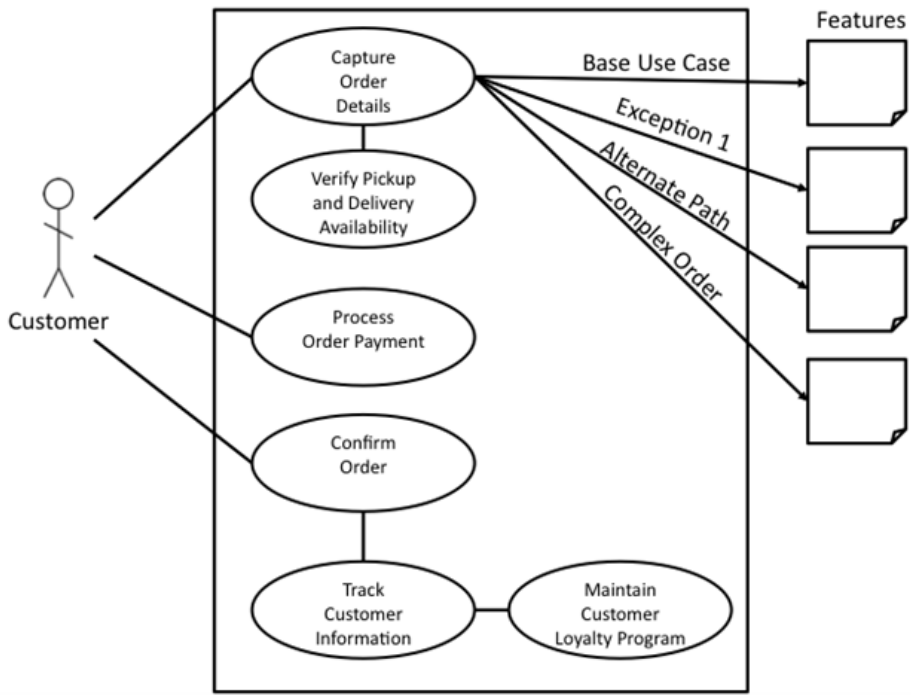


Figure 6-2. Creating Features from a Use Case

6.2.3 Define Software Project Activities: Outputs

The outputs in Section 6.2.3 of the *PMBOK® Guide* are applicable to defining software project activity outputs with the modifications below. In addition to these, the outputs described in 6.2.3.4 and 6.2.3.5 also apply to software project activities.

6.2.3.1 Activity List

See also Section 6.2.3.1 of the *PMBOK® Guide*. Activity lists for software projects may include coordination with entities external to the software project. The software development team may need access to testing facilities, infrastructure equipment, and access to multiple user environments. These project elements may be outside the control of the software project manager and may require external scheduling to avoid a negative impact to the software project schedule.

6.2.3.2 Activity Attributes

See also Section 6.2.3.2 of the *PMBOK® Guide*. Attributes that may be included for activities on a software project activity list include but are not limited to the following:

- 0Dependencies and enabling precedent activities,
- 0Stakeholder value or priority,
- 0Estimated effort, size, complexity, and/or risk,
- 0Security and/or safety standards and constraints,
- 0Special competency requirements for staff resources,
- 0Required resources for development, testing, and validating of equipment, and
- 0Class of service for on-demand scheduling.

6.2.3.3 Milestone List

See also Section 6.2.3.3 of the *PMBOK® Guide*. Milestones for software projects are defined in various ways for various development environments and project life cycles. For example, some software project life cycles (e.g., the Rational Unified Process) define anchor points, which are points in time when major project life cycle phase transitions occur.

In incremental software development, milestones may be set to denote architectural design reviews, customer delivery, and acceptance points, or as feature set accomplishments. Often each increment includes a completion or validation milestone, but that is not required.

On-demand scheduling methods do not usually have specific milestones; progress is measured by customer satisfaction within the time-to-complete cadence. Calendar-based coordination conferences may be held to discuss performance, but these are rarely associated with a specific goal or technical criterion.

A useful technique for reducing project risk is to define joint milestones with interdependent projects such as hardware procurement, installation and configuration for platform and environment delivery. Identified risks can then be mitigated. This type of program/ portfolio management is often critical to the successful delivery of software products especially in a constrained schedule.

6.2.3.4 Safety and Security Analyses

Public safety and cyber security issues may impact the sequencing of some software activities to satisfy requirements and standards during a software project.

6.3 Sequence Software Project Activities

According to Section 6.3 of the *PMBOK® Guide,* Sequence Activities is the process of identifying and documenting relationships among the project activities. Sequencing activities for software projects differ somewhat to those in Section 3 of the *PMBOK® Guide* because the sequencing methods used may be based on customer value, technical risk, technical architecture and design, specific expertise availability, as well as other technical and staffing dependencies.

Dependencies in database structure, infrastructure needs and other architecture and design concerns exist in many software projects. However, for a new application domain or a large, complex software project in a new or existing domain, there is often a need to establish and refine the operational concepts, build prototypes, and/or define an architecture or infrastructure before specifying the functional requirements for the product. How much time is needed for these activities and how concurrently they can be accomplished depends on the familiarity, size, and complexity of the product; and on the risk profiles; and, in particular, on the likelihood of changes in the product requirements.

Architecture has a significant impact on sequencing in several areas. First, the actual architecture development is not easily estimated, and therefore, any software directly related to the architectural design may need to be delayed until the architecture is complete. In some instances, only some of the architectural decisions are unknown, so early investment in activities that prove the effectiveness of an architectural solution or build an initial architectural structure may be effective. These activities may be called firing tracer rounds or developing walking skeletons or steel threads, but the intent is to make sure that the key architectural decisions are feasible and that the solutions are available for functional development. For example, exception handling, data assurance, and security patterns need to be established early for consistency across the system. Second, the architecture supports the ability to define pieces of the product that can be tested and the efficacy of mockups, stubs, or dummies that allow testing of incomplete functionality. This work needs to precede the development so that testing techniques such as test driven development, have a framework to grow around. This is of particular significance in larger systems that integrate with outside or legacy systems beyond the project manager's control.

In a similar way, the nonfunctional and quality requirements impact the sequencing of activities by requiring time to implement cross-cutting strategies (such as error-handling, function criticality, failure modes). The need for certification of software components due to regulatory, safety, or security requirements may also affect sequencing due to the overhead costs of certification. It is usually more cost-effective to bundle changes to certified code so that recertification activities are not repeated unnecessarily.

Software schedules are frequently revised. In-process prototyping and coding experimentation may be needed to support decision making. These kinds of activities may not be identified during initial scheduling, so the ripple effects they cause can impact sequencing in real time. Needed rework to fix defects is another sequencing issue. Activities that were not anticipated but are still necessary for project completion often arise. This unanticipated work (sometimes referred to as *dark matter*) can often take precedence over other work and is sometimes tracked independently. Section 6.7 discusses the roles burn-down and burn-up charts in relation to these issues.

Adjustments to schedule sequencing for adaptive life cycle software projects is more dynamic and occurs more frequently than for predictive life cycle projects, and generally provides more opportunities to absorb unscheduled work. A plan is created to provide structure and focus for the adaptive iterations, their content, and any product release points. However, the plan is revisited often to incorporate changes related to business feedback based on factors such as demonstrations of the evolving product, productivity (velocity) data gathered during development, unscheduled work, and retrospective findings.

Adaptive software projects usually sequence activities prior to beginning iterative development, but the scope of this initial sequencing may be vague and will be refined as the project progresses. In some cases, higher levels in feature and story breakdowns are used to coordinate lower levels—with the unscheduled work absorbed into the estimates for the higher-level deliverables.

On-demand scheduling techniques, and some highly adaptive life cycles, allow the work to flow to whatever suitable staff resources become available. This is sometimes referred to

as *late binding* of the work to the resources. However, the order of sequence execution is nondeterministic until the actual time of binding the work item to the available resource or resources. The available staff resource, or resources, dynamically selects the next work to be done based on the value of the queued work activities. Value is defined by project specific risks or constraints (e.g., cost of delay, value to customer, class of service, or criticality).

Rather than a date-certain schedule of events or a specified increment where a certain number of tasks are in scope to be completed, on-demand techniques establish a regular cadence of releases (or other events). The timing of the cadence is determined through measures such as velocity or a statistics-based typical lead or transit time for a task. The cadence then provides an indication of how long a customer or software project manager can expect to wait for a particular activity or task to complete. Work-in-progress limits are used to maintain resource viability and even out flow, and are adjusted according to statistical measures maintained throughout the process. There may be a visual indicator (a workflow chart) to provide broad visibility and help to identify and resolve bottlenecks or make better use of available resources.

6.3.1 Sequence Software Project Activities: Inputs

The inputs for sequencing software project activities in Section 6.3.1 of the *PMBOK® Guide* are generally applicable for sequencing software project activities, with the modification of 6.3.1.6 and the additional inputs in of 6.3.1.8 and 6.3.1.9.

See Section 6.3.1.1 of the PMBOK® Guide.6.3.1.1 Schedule Management Plan

See Section 6.3.1.1 of the *PMBOK® Guide*.

6.3.1.2 Activity List

See Section 6.3.1.2 of the *PMBOK® Guide*.

6.3.1.3 Activity Attributes

See Section 6.3.1.3 of the *PMBOK® Guide*.

6.3.1.4 Milestone List

See Section 6.3.1.4 of the *PMBOK® Guide*.

6.3.1.5 Project Scope Statement

See Section 6.3.1.5 of the *PMBOK® Guide*.

6.3.1.6 Enterprise Environmental Factors

See Section 6.3.1.6 of the *PMBOK® Guide*.

6.3.1.7 Organizational Process Assets

Governance models often have set anchor points, templates, and environmental issues that can impact the scheduling sequence for software projects. The enterprise architecture, if there is one, may also impact the sequencing. Organizational parameters for valuing

software investments may be of use in identifying the value of functionality to the customer and thus impact sequencing.

6.3.1.8 Architectural and IV&V Constraints

Architecture and IV&V plans can significantly impact the sequencing when there are constraints on the functionality needed to verify and validate cross functional, multi-system, multi-platform, or multi-environment requirements (See Section 6.3).

6.3.1.9 Safety and Security Analyses

Public safety and cyber security issues may impact the sequencing of some software activities to meet requirements and standards. Certification activities are always expensive, so the sequencing should try to minimize the number of certification cycles.

6.3.2 Sequence Software Project Activities: Tools and Techniques

The tools and techniques for sequencing project activities in Section 6.3.2 of the *PMBOK® Guide* are applicable to sequencing software project activities. In addition to these, the tools and techniques in 6.3.2.4 through 6.3.2.8 are applicable to sequencing software project activities.

6.3.2.1 Precedence Diagramming Method

See Section 6.3.2.1 of the *PMBOK® Guide*.

6.3.2.2 Dependency Determination

See Section 6.3.2.1 of the *PMBOK® Guide*.

6.3.2.3 Leads and Lags

See Section 6.3.2.1 of the *PMBOK® Guide*.

6.3.2.4 Portfolio Management Approaches

See Section 1.4.1 of this software extension.

6.3.2.5 SAIV and Time Boxing

Using SAIV (Schedule As Independent Variable) for sequencing software project activities can help to ensure that the most valuable features or functionality are available when time is exhausted because the most important features have been implemented. The SAIV concept is applied in a variety of techniques, including time boxing and date-certain scheduling. As shown in Figure 6-3, product scope can be varied varying when cost and time have been set. SAIV can be applied within increments, releases, or for complete products.

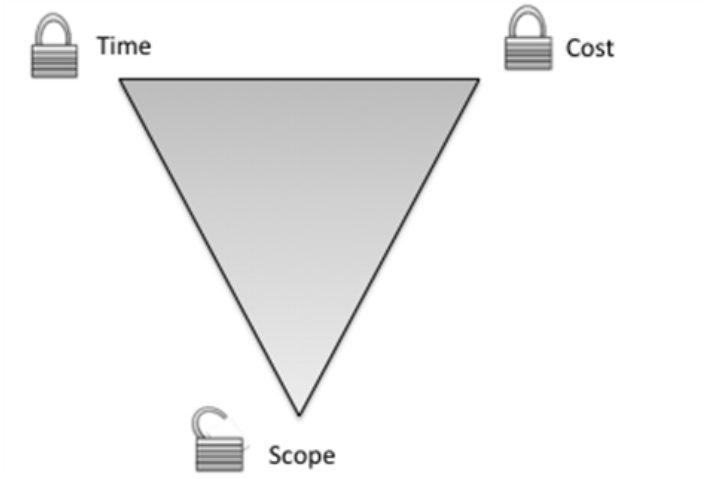


Figure 6-3. Schedule as Independent Variable

This method depends on the ability to set relative values on items of work such as features or stories. These values may change over time, but adaptive life cycles allow for that change by frequently reassessing value. This presupposes that the customer or other stakeholders are either available or are represented by surrogates whenever values are assessed.

SAIV can also be applied to exploratory work done when determining architectures and designs. One method is to define the hypothesis that needs to be proven, and a time box within which the proof must be completed. It is important to create a clear hypothesis (for example using a statement or if-then format) from which to design the experiment.

6.3.2.6 Work in Progress Limits and Classes of Service

See on-demand scheduling in Section 6.1 of this extension.

6.3.2.7 Feature Sets

A feature set specifies an aggregate of functionality that delivers business value and may include work activities in addition to functions. Different software development methods use different terminology to refer to features. It is up to the software project manager and the project team to decide which terminology to use.

As shown in Figure 6-4, by identifying feature sets, the team designs, builds, and delivers all the stories within a feature before moving on to the next feature. This is an important structure for feedback purposes. Even when the same code base is impacted by more than one feature, in feature sets, the team does all the work for one feature, tests, and remediates; then the next feature, tests and remediates; and so on, until the planned scope is accomplished or the increment is completed. This incurs a cost to development for a return in reduced testing complexity and deferring the commitment of design and functionality until the next feature is built.

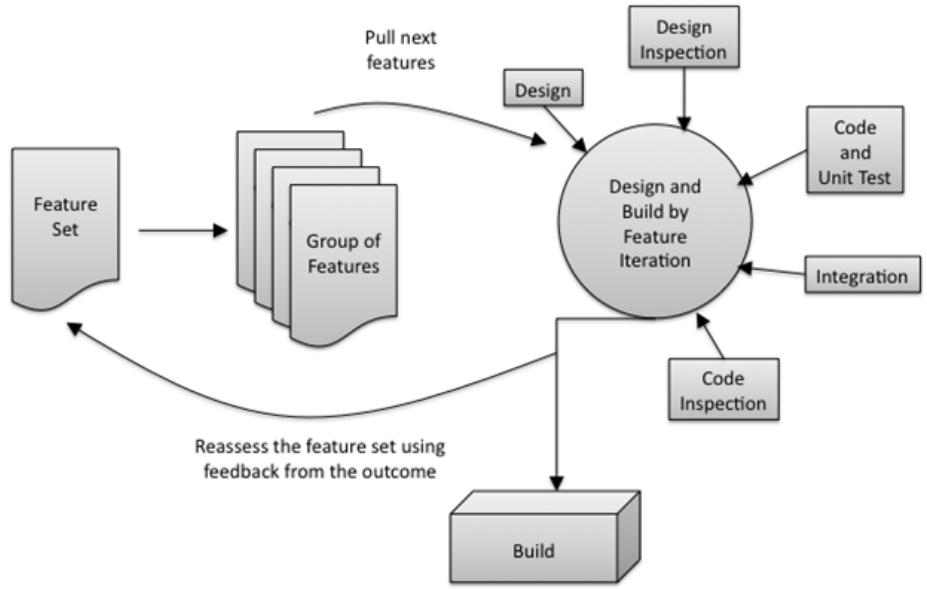


Figure 6-4. Scheduling Using Feature Sets

6.3.2.8 Service Level Agreements

There may be an agreement between the developer and the customer or other stakeholder that sets the amount of work to be accomplished expected from an organization over a specified period of time. This establishes the required development capacity and may impact sequencing.

6.3.3 Sequence Software Project Activities: Outputs

The outputs for project sequencing in the *PMBOK® Guide* are applicable for sequencing software project activities. In addition to these, the outputs specific to software project sequencing activities are provided in 6.3.3.3 through 6.3.3.5.

6.3.3.1 Project Schedule Network Diagrams

See Section 6.3.3.1 of the *PMBOK® Guide*.

6.3.3.2 Project Documents Updates

See Section 6.3.3.2 of the *PMBOK® Guide*.

6.3.3.3 Features Sets

See Section 6.3.2.8 of this software extension.

6.3.3.4 Release Plans

A release plan lays out the overall project schedule in iterative stages for the delivery of functionality for customer evaluation and/or delivery into the users' environment. The release plan is highly dependent upon the velocity of the software development team.

Comparing estimated velocity to the actual time it takes to accomplish the work over a

period of several iterations provides a baseline for estimating the time for future releases.

6.3.3.5 Architectural and Nonfunctional Dependencies

Architectural and nonfunctional dependencies need to be updated to reflect the planned project activities to prevent duplication of work or rework by other project teams or initiatives.

6.4 Estimate Software Project Activity Resources

According to Section 6.4 of the *PMBOK® Guide* estimating resources for project activities is the process of estimating the type and quantities of material, people, equipment, or supplies required to perform each activity.

Because software is developed by the intellectual efforts of software developers, software projects are heavily dependent upon human resources more than any other kind of software project resource. The skills and abilities of the software developers is a significant factor in estimating the number of software developers needed. Studies have shown as much as 26:1 variations in individual software developer productivity, and 10:1 variations are not uncommon even among software developers having similar education and work experience **[20]**.

The process of determining the roles required for the project can be determined by reviewing the product requirements, the project objectives, stakeholder's goals, and budget and schedule constraints. As a software project evolves, requirements will be refined, features and user stories will be identified, and the human resources needed to satisfy the project goals are checked against the current team's collective skills. Gaps may indicate that additional or different roles are required. Likewise, productivity (velocity) and quality metrics may provide insights into team role requirements as the project progresses. In some cases, the software project manager may be given a collection of team members without the opportunity to identify needed project roles or to adjust them as the project evolves. In other cases, the project manager may be asked for the kinds of roles that need to be filled, the timing for the roles, and in what quantities.

Other resource requirements for software projects may include resources for architectural studies and several kinds of support activities (e.g., configuration management, quality assurance, documentation, user training). Test facilities, software for performance testing and multi-configuration test suites, and multiple target environments or platforms for deployment are examples of other software resources that may be required.

6.4.1 Estimate Software Project Activity Resources: Inputs

As stated in 6.4, software developers are the most important resources for a software project. Because software productivity varies widely among software developers (even among those having similar educations and work experiences) historical data concerning productivity at the team and individual levels is a valuable input for estimating software project activity resources, in addition to the inputs described in Section 6.4.1 of the *PMBOK® Guide*. Software project managers who use adaptive life cycles have the opportunity to collect productivity data on a frequent, ongoing basis and may be able to adjust human resources as the project progresses. Personnel adjustments are typically more difficult for predictive life cycle software projects.

Other inputs for estimating software project activity resources involve using a results chain and other forms of analyses to identify key assumptions and resources outside of the software development activities for a software project

The inputs in Section 6.4.1 of the *PMBOK® Guide* are applicable inputs for estimating

software project activity resources.

6.4.1.1 Schedule Management Plan

See Section 6.4.1.1 of the *PMBOK® Guide*.

6.4.1.2 Activity List

See Section 6.4.1.2 of the *PMBOK® Guide*.

6.4.1.3 Activity Attributes

See Section 6.4.1.3 of the *PMBOK® Guide*.

6.4.1.4 Resource Calendars

See Section 6.4.1.4 of the *PMBOK® Guide*.

6.4.1.5 Risk Register

See Section 6.4.1.5 of the *PMBOK® Guide*.

6.4.1.6 Activity Cost Estimates

See Section 6.4.1.6 of the *PMBOK® Guide*.

6.4.1.7 Enterprise Environmental Factors

See Section 6.4.1.7 of the *PMBOK® Guide*.

6.4.1.8 Organizational Process Assets

See Section 6.4.1.8 of the *PMBOK® Guide*.

6.4.2 Estimate Software Project Activity Resources: Tools and Techniques

The tools and techniques for estimating project activity resources in the *PMBOK® Guide* are applicable for software projects. In addition to these, the tools and techniques in 6.4.2.6 and 6.4.2.7 apply to estimating software project activity resources.

6.4.2.1 Expert Judgment

See Section 6.4.2.1 of the *PMBOK® Guide*.

6.4.2.2 Alternative Analysis

See Section 6.4.2.2 of the *PMBOK® Guide*.

6.4.2.3 Published Estimating Data

See Section 6.4.2.3 of the *PMBOK® Guide*.

6.4.2.4 Bottom-Up Estimating

See Section 6.4.2.4 of the *PMBOK® Guide*.

6.4.2.5 Project Management Software

See Section 6.4.2.5 of the *PMBOK® Guide*.

6.4.2.6 Service Level Agreements

There may be an agreement between the developer and the customer or other stakeholder that sets the amount of work to be accomplished expected from an organization over a specified period of time. This establishes the required development capacity and may impact resource estimation.

6.4.2.7 Other Tools and Techniques

Other tools and techniques for estimating software project activity resources include use of algorithmic estimation models and function point/story point/use-case estimation tools.

6.4.3 Estimate Software Project Resources: Outputs

The outputs listed in Section 6.4.3 of the *PMBOK® Guide* are applicable for estimating software project resources.

6.4.3.1 Activity Resource Requirements

See Section 6.4.3.1 of the *PMBOK® Guide*.

6.4.3.2 Resource Breakdown Structure

See Section 6.4.3.2 of the *PMBOK® Guide*.

6.4.3.3 Project Documents Updates

See Section 6.4.3.3 of the *PMBOK® Guide*.

6.5 Estimate Software Project Activity Duration

The difficulty in estimating software project activity durations is the result of many factors: fundamental intangibility of software, broad variance in productivity of software professionals, need for change to meet emergent requirements, often unprecedented nature of the software product, unknown competencies of staff resources or product team, unknown existing hardware or software defects, and need to incorporate legacy software, commercial software, customer-supplied software, or open-source software into the software product. Even if these factors are taken into consideration, the result may be accurate for the known work, but cannot account for the unidentified, unknown work that will need to be performed.

Exacerbating the estimation challenges in software projects is the nonlinear nature of the

scaling of software work; a product twice as big or twice as complex, however measured, typically requires more than twice as much work, and more than twice as much time because of the interdependencies of the work activities and the increased communication needs of the software developers. Adding additional work activities may result in significant delays in the delivery of each increment of value and may result in schedule perturbations, which future complicates the ability to accurately update estimated durations. The software project life cycle and scheduling method used should account for the significant risk of the likely estimation error.

Because effort is the product of people and time, the schedule durations of software project activities depend on estimated effort and personnel resources available. Section 7.2.2 provides information on additional ways to estimate effort for software projects.

6.5.1 Estimate Software Project Activity Durations: Inputs

The inputs in Section 6.5.1 of the *PMBOK® Guide* can be used to estimate software project activity durations. They are the schedule management plan, the activity list, activity attributes, activity resource requirements, resource calendars, project scope statement, risk register, resource breakdown structure, enterprise environmental factors, and organizational process assets. See Section 6.5.1.1 – 6.5.1.10 of the *PMBOK® Guide*.

6.5.1.1 Schedule Management Plan

See Section 6.5.1.1 of the *PMBOK® Guide*.

6.5.1.2 Activity List

See Section 6.5.1.2 of the *PMBOK® Guide*.

6.5.1.3 Activity Attributes

See Section 6.5.1.3 of the *PMBOK® Guide*.

6.5.1.4 Activity Resource Requirements

See Section 6.5.1.4 of the *PMBOK® Guide*.

6.5.1.5 Resource Calendars

See Section 6.5.1.5 of the *PMBOK® Guide*.

6.5.1.6 Project Scope Statement

See Section 6.5.1.6 of the *PMBOK® Guide*.

6.5.1.7 Risk Register

See Section 6.5.1.7 of the *PMBOK® Guide*.

6.5.1.8 Resource Breakdown Structure

See Section 6.5.1.8 of the *PMBOK® Guide*.

6.5.1.9 Enterprise Environmental Factors

See Section 6.5.1.9 of the *PMBOK® Guide*.

6.5.1.10 Organizational Process Assets

See Section 6.5.1.10 of the *PMBOK® Guide*.

6.5.1.11 Additional Inputs

In addition to those presented in Section 6.5.1 of the *PMBOK® Guide* lists of activities, features organized as lists or groups, and customer stories are useful inputs for estimating software project activity durations. Velocity and rework metrics are also useful inputs.

6.5.2 Estimate Software Project Activity Durations: Tools and Techniques

The tools and techniques presented in Section 6.5.2 of the *PMBOK® Guide* are applicable for estimating software project activity durations.

6.5.2.1 Expert Judgment

See Section 6.5.2.1 of the *PMBOK® Guide*.

6.5.2.2 Analogous Estimating

See Section 6.5.2.2 of the *PMBOK® Guide*.

6.5.2.3 Parametric Estimating

See Section 6.5.2.3 of the *PMBOK® Guide*.

6.5.2.4 Three-Point Estimating

See Section 6.5.2.4 of the *PMBOK® Guide*.

6.5.2.5 Group Decision-Making Techniques

See Section 6.5.2.5 of the *PMBOK® Guide*.

6.5.2.6 Reserve Analysis

See Section 6.5.2.6 of the *PMBOK® Guide*.

6.5.2.7 Service Level Agreements

There may be an agreement between the developer and the customer or other stakeholder that sets the amount of work to be accomplished expected from an organization over a specified period of time. This establishes the required development capacity and may impact sequencing.

6.5.3 Estimate Software Project Activity Durations: Outputs

The outputs for estimating project activity durations are applicable for software projects.

6.5.3.1 Activity Duration Estimates

See Section 6.5.3.1 of the *PMBOK® Guide*.

6.5.3.2 Project Documents Updates

See Section 6.5.3.2 of the *PMBOK® Guide*.

6.6 Develop Software Project Schedule

The form of the software project schedule may be the same as described in Section 6.5 of the *PMBOK® Guide* or it may take a different form altogether. In addition to the approach described in the *PMBOK® Guide*, a more flexible approach facilitates the expected changes that inevitably occur in a software project schedule. Software project schedules and plans change, driven by customer requests, project feedback, and by the emergence of previously unidentified work activities. The format of a software project schedule may be unfamiliar to some stakeholders. Instead of using a network diagram, a prioritized backlog of work may be the preferred method for illustrating and managing the sequence of project activities. Tools and deliverables that are easy to change are often preferred instead of the harder-to-update network diagrams.

The approach of maintaining a prioritized backlog of work activities is similar to rolling wave planning, where a top-level schedule is completed for the entire project and only the proximate elements of the schedule are completed in detail as the project evolves.

6.6.1 Develop Software Project Schedule: Inputs

The inputs for developing a project schedule in the *PMBOK® Guide* are applicable for developing a software project schedule. In addition to these, an additional input is listed in 6.6.1.14.

6.6.1.1 Schedule Management Plan

See Section 6.6.1.1 of the *PMBOK® Guide*.

6.6.1.2 Activity List

See Section 6.6.1.2 of the *PMBOK® Guide*.

6.6.1.3 Activity Attributes

See Section 6.6.1.3 of the *PMBOK® Guide*.

6.6.1.4 Project Schedule Network Diagrams

See Section 6.6.141 of the *PMBOK® Guide*.

6.6.1.5 Activity Resource Requirements

See Section 6.6.1.5 of the *PMBOK® Guide*.

6.6.1.6 Resource Calendars

See Section 6.6.1.6 of the *PMBOK® Guide*.

6.6.1.7 Activity Duration Estimates

See Section 6.6.1.7 of the *PMBOK® Guide*.

6.6.1.8 Project Scope Statement

See Section 6.6.1.8 of the *PMBOK® Guide*.

6.6.1.9 Risk Register

See Section 6.6.1.9 of the *PMBOK® Guide*.

6.6.1.10 Project Staff Assignments

See Section 6.6.1.10 of the *PMBOK® Guide*.

6.6.1.11 Resource Breakdown Structure

See Section 6.6.1.11 of the *PMBOK® Guide*.

6.6.1.12 Enterprise Environmental Factors

See Section 6.6.1.12 of the *PMBOK® Guide*.

6.6.1.13 Organizational Process Assets

See Section 6.6.1.13 of the *PMBOK® Guide*.

6.6.1.14 Additional Inputs

Additional inputs for developing a software project schedule include activity lists, features and feature sets, and stories. Other inputs include historical data on project team cadence and velocity, and service level agreements for on-demand scheduling.

6.6.2 Develop Software Project Schedule: Tools and Techniques

The tools and techniques for developing a project schedule are applicable to developing a software project schedule, with the modification of 6.6.2.8. In addition to those in Section 6.6.2 of the *PMBOK® Guide,* the tools and techniques described in 6.6.2.7 and 6.6.2.9 are applicable when developing a software project schedule:

6.6.2.1 Schedule Network Analysis

See Section 6.6.2.1 of the *PMBOK® Guide*.

6.6.2.2 Critical Path Method

See Section 6.6.2.2 of the *PMBOK® Guide*.

6.6.2.3 Critical Chain Method

See Section 6.6.2.3 of the *PMBOK® Guide*.

6.6.2.4 Resource Optimization Techniques

See Section 6.6.2.4 of the *PMBOK® Guide*.

6.6.2.5 Modeling Techniques

See Section 6.6.2.5 of the *PMBOK® Guide*.

6.6.2.6 Leads and Lags

See Section 6.6.2.6 of the *PMBOK® Guide*.

6.6.2.7 Schedule Compression for Software Projects

Compressing the schedule of a software project results in lower technical quality and/or a nonlinear increase in effort because more people will be needed to meet the schedule and the number of communication paths among increased numbers of project members increases exponentially (on the order of *n*-squared, where *n* is the number of project members). The non-linearity in staffing thus results from the increased communication and coordination that occur when a software development team becomes larger. A well-known rule of thumb states that software projects rarely succeed if the project schedule is compressed more than 25%, regardless of the number of people added to the project. And, the well-known Brooks Law states "adding manpower to a late project makes it later." **[21]**

6.6.2.8 Scheduling Tool

See Section 6.6.2.8 of the *PMBOK® Guide*.

6.6.2.9 Release and Iteration Planning for Software Projects

Release and iteration planning processes for adaptive software project life cycles are used to develop the sequencing of work activities to be accomplished based on the ordering of deliverable increments of software. During release planning, the project schedule is ordered by the priority of the deliverables and is roughly divided up into release cycles, such as weekly, monthly, or quarterly. This determines the estimated number of iterative cycles. Unanticipated work is by definition unknowable at the beginning of a project. However, its existence is almost a certainty. When planning for a software release, the likelihood of unanticipated work activities should be taken into account.

The adaptive life cycle team then plans the work activities for the next iterative cycle in just enough detail to accomplish the work, typically with daily reviews of the work accomplished. By working directly with customers or other stakeholders, the software developers can readily understand their needs and develop, build, and deliver the software efficiently and effectively. Using this method of planning, the anticipated unknowns may

indicate that the selected increments are too large for delivery within the required release cycle. When this occurs, the team partitions the functionality to deliver what can be delivered, which requires an adjustment in the prioritization of work activities and the product backlog.

6.6.3 Develop Software Project Schedule: Outputs

The outputs from developing a project schedule are applicable to software projects. In addition to these, the output in 6.6.3.9 applies to developing a software project schedule.

6.6.3.1 Schedule Baseline

See Section 6.6.3.1 of the *PMBOK® Guide*.

6.6.3.2 Project Schedule

See Section 6.6.3.2 of the *PMBOK® Guide*.

6.6.3.3 Schedule Data

See Section 6.6.3.3 of the *PMBOK® Guide*.

6.6.3.4 Project Calendars

See Section 6.6.3.4 of the *PMBOK® Guide*.

6.6.3.5 Project Management Plan Updates

See Section 6.6.3.5 of the *PMBOK® Guide*.

6.6.3.6 Project Documents Updates

See Section 6.6.3.6 of the *PMBOK® Guide*.

6.6.3.7 Release and Iteration Plan Updates

Release and iteration plans and updates to those plans are additional outputs from developing a schedule for an adaptive life cycle software project.

6.7 Control Software Project Schedule

Controlling a software project schedule is a challenging proposition. To evaluate schedule variance, the software project manager needs to understand the technical variance as well. Technical variance in software can be as intangible as the software itself. For this reason, measures and reviews need to be focused and evidence-based.

For adaptive life cycle software projects, the rate of work activity completion (i.e., velocity), results from retrospective reviews and customer feedback from periodic demonstrations of the evolving software project are used to adapt the upcoming release and iteration plans to incorporate mid-course adjustments and align the project with stakeholder objectives. This may include reprioritization of the backlog of remaining work (e.g., stories), changing the development team's work processes, or adjusting the

engagement model with the customer. As the project progresses more emphasis should be placed on emerging velocity trends as an indicator of the final completion date.

Establishing stable software development teams and limiting the amount of work in process can significantly reduce the variables associated with schedule control. Therefore, in addition to reprioritization and refactoring of the remaining work, schedule control often involves making changes to team structures and managing the flow of work through the teams.

6.7.1 Control Software Project Schedule: Inputs

The inputs for controlling a project schedule in Section 6.7.1 of the *PMBOK® Guide* are applicable to controlling software project schedules, with the modification of 6.7.1.3.

6.7.1.1 Project Management Plan

See Section 6.7.1.1 of the *PMBOK® Guide*.

6.7.1.2 Project Schedule

See Section 6.7.1.2 of the *PMBOK® Guide*.

6.7.1.3 Work Performance Data for Software Projects

The cadence of recent work completion, current velocity metrics, and service-level agreements for on-demand scheduling can provide work performance data for adaptive life cycle software projects.

6.7.1.4 Project Calendars

See Section 6.7.1.4 of the *PMBOK® Guide*.

6.7.1.5 Schedule Data

See Section 6.7.1.5 of the *PMBOK® Guide*.

6.7.1.6 Organizational Process Assets

See Section 6.7.1.6 of the *PMBOK® Guide*.

6.7.2 Control Software Project Schedule: Tools and Techniques

The tools and techniques presented in Section 6.7.2 of the *PMBOK® Guide* for controlling a project schedule are applicable to controlling a software project schedule with the following modification of 6.7.2.6 and the addition of 6.7.2.8 through 6.7.2.14.

6.7.2.1 Performance Reviews

In many software methods, performance reviews (as described in the PMBOK® Guide) are part of the technical review cycle. In most cases, the best measurement of performance is the value created/delivered over time. However, care should be taken when reviewing software performance to ensure that all the ancillary activities that are not specifically related to the software development are progressing as well. Ancillary activities include

infrastructure, testing environments, test cases, interface control, configuration management, equipment or supply acquisition, deployment planning, and logistics activities.

6.7.2.2 Project Management Software

See Section 6.7.2.2 of the *PMBOK® Guide*.

6.7.2.3 Resource Optimization Techniques

See Section 6.7.2.3 of the *PMBOK® Guide*.

6.7.2.4 Modeling Techniques

See Section 6.7.2.4 of the *PMBOK® Guide*.

6.7.2.5 Leads and Lags

See Section 6.7.2.5 of the *PMBOK® Guide*.

6.7.2.6 Schedule Compression

As described in Section 6.6.2.6 of this software extension, compressing a software project schedule more than 25% by increasing staff results in a nonlinear increase in schedule thus making the project later. The only possible way to compress a software schedule is to remove features, thus reducing the work to be accomplished.

6.7.2.7 Scheduling Tool

See Section 6.7.2.7 of the *PMBOK® Guide*.

6.7.2.8 Variance Analysis

As stated in Section 6.7.1.3, cadence of recent work completion, current velocity metrics, and service-level agreements for on-demand scheduling can indicate variances in work performance data when using an adaptive software project life cycle.

6.7.2.9 Evidence-Based Anchor Point Milestone and Architectural Reviews

These technical project reviews evolved from early work at Bell Labs and have been recommended in various standards (including ISO/IEC/IEEE 12207, ISO/IEC/IEEE Standard 1028, USC MBASE and the Rational Unified Process). They suggest the following guidelines:

- 0Base reviews on *evidence* (e.g. demonstrations of working software, simulations) provided by the developer and validated by independent experts. A laundry list of disassociated work products completed is not evidence.
- 0Develop a feasibility description to integrate the evidence.
- 0Provide evidence to indicate that if the system is built to the specified architecture, it will:
    - Satisfy the requirements—that is, capability, interfaces, level of service, and evolution;
    - Support the operational concept;
    - Be buildable within the budgets and schedules in the plan;
    - Generate a viable return on investment; and
    - Generate satisfactory outcomes for all of the success-critical stakeholders.

- 0Resolve all major risks or cover in risk management plan.
- 0Utilize to serve as basis for stakeholders' commitment to proceed.

6.7.2.10 Retrospectives

Retrospectives are a variant of the performance reviews listed in Section 6.7.2.1 in the *PMBOK® Guide* but they are typically held more frequently than traditional performance reviews—usually after each development iteration.

6.7.2.11 Cumulative Flow Diagrams

Cumulative flow diagrams (CFDs) (see Figure 6-5) provide a simple method of tracking work-in-progress and visually analyzing the trend line for the projected delivery of implemented features. CFDs provide metrics that allows teams and managers to react early to developing problems and, in addition they provide visibility into the overall project life cycle. Because CFDs plot both the total product scope and the progress of individual work items, they visually communicate progress as well as the proportion of total completeness.

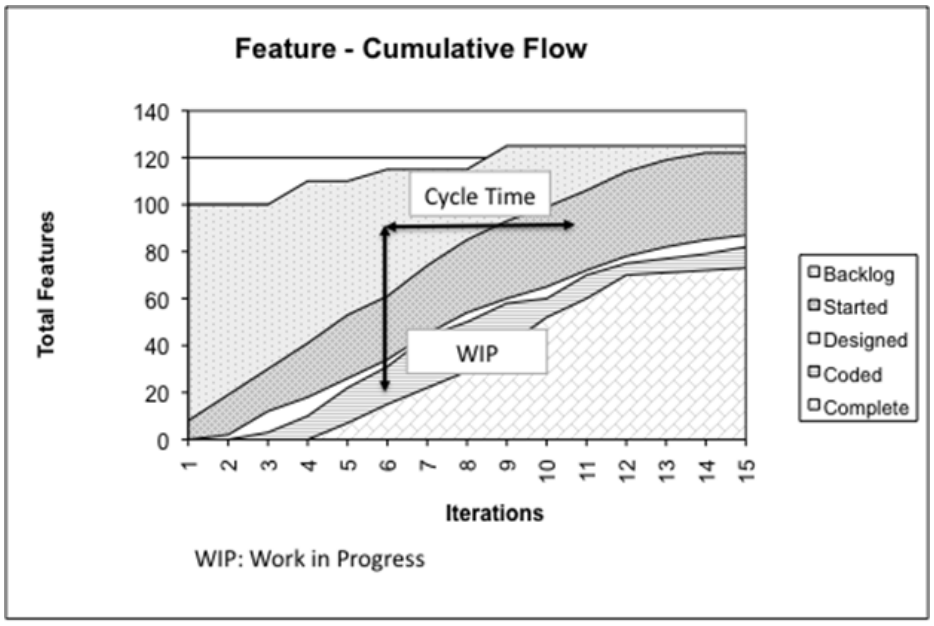


Figure 6-5. A Cumulative-Flow Diagram for Tracking Product Features

6.7.2.12 Workflow Board with Daily Walkthrough

A workflow board is a visual depiction of work flowing through a software project using an on-demand scheduling approach. Daily walkthroughs provide immediate feedback on blockages and resource issues for the entire team, effectively supporting decisions.

6.7.2.13 Reprioritization Reviews

Reprioritization reviews are elements of the iterative scheduling process. Project progress may require adjustment of priorities among planned work activities.

6.7.2.14 Burn-up and Burn-down Charts

A burn-up or burn-down chart visually illustrates the accomplishments of the software team

in their delivery of business value as measured by completed features, stories, or other work units. A release burn-up chart is illustrated in Figure 6-6.

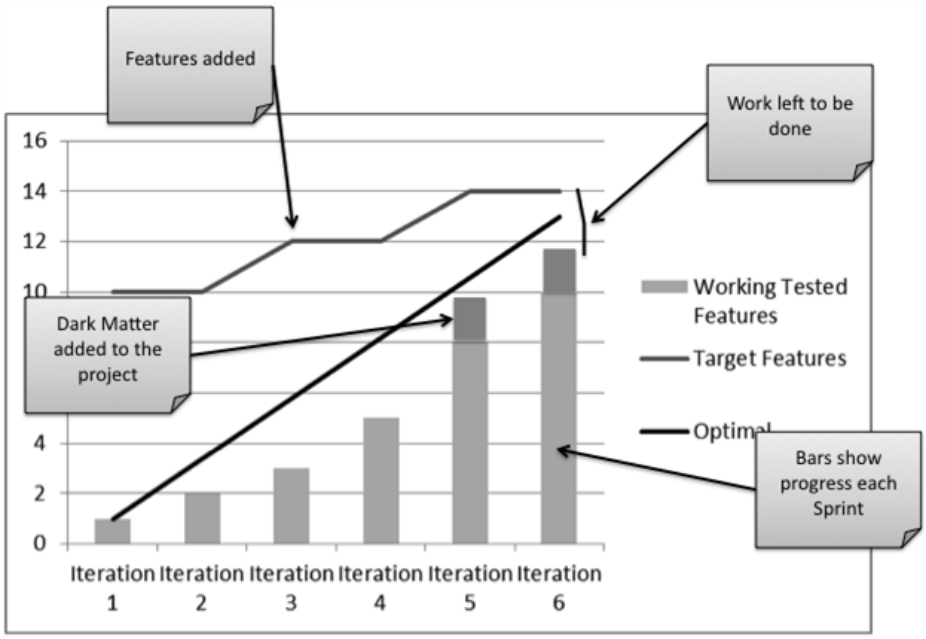


Figure 6-6. Release Burn-up Chart

6.7.3 Control Software Project Schedule: Outputs

The outputs for controlling a project schedule in Section 6.7.3 of *PMBOK® Guide* are applicable for controlling a software project schedule, with the addition of 6.7.3.7.

6.7.3.1 Work Performance Information

See Section 6.7.3.1 of the *PMBOK® Guide*.

6.7.3.2 Schedule Forecasts

See Section 6.7.3.2 of the *PMBOK® Guide*.

6.7.3.3 Change Requests

See Section 6.7.3.3 of the *PMBOK® Guide*.

6.7.3.4 Project Management Plan Updates

See Section 6.7.3.4 of the *PMBOK® Guide*.

6.7.3.5 Project Documents Updates

See Section 6.7.3.5 of the *PMBOK® Guide*.

6.7.3.6 Organizational Process Assets Updates

See Section 6.7.3.6 of the *PMBOK® Guide*.

6.7.3.7 Additional Outputs

Release and iteration plan updates and velocity measures are useful outputs for adaptive life cycle software projects, as are service level agreement adjustments for on-demand scheduling.

7 Software Project Cost Management

As stated in the introduction to Section 7 of the *PMBOK® Guide*, Project Cost Management includes the processes involved in estimating, budgeting, funding, managing, and controlling costs so that the project can be completed within the approved budget. This section of the *Software Extension to the PMBOK® Guide – Fifth Edition* discusses this topic from a software project perspective.

A large corporation or government agency may develop many new software products and modify hundreds of existing products every year. As a result, software cost estimating is now a mainstream activity for every organization that builds software; it has become a critical process for those organizations. The effort required to develop or modify software is almost entirely dependent on the skills and abilities of individual team members, the interactions among team members, technical leadership direction, and the culture and processes in the software development environment. Cost estimation for software projects may include identifying and forecasting the cost of maintaining and evolving a software product and licensing or updating commercially acquired components over many years, in addition to making estimates for developing or modifying software. Estimating with this amount of variability is difficult even when a significant amount of historical data exists.

Effort and schedule are closely related for software projects. Because staff-hours are the primary cost factor for software development, effort estimation is the basis for estimating the cost of a software project. Section 6 of this software extension provides guidance on managing the relationship between effort estimation and schedule estimation for software projects. This section addresses effort estimation, in addition to other aspects of managing software project cost, to ensure that software project managers understand the impact of the variability of software cost drivers on software project cost estimates. When using adaptive life cycle models, which maintain flexibility as late into the development process as possible, project managers still need to estimate the effort and cost of the project. However, the volatility of project attributes, such as rapidly evolving technology, changing and emerging architecture and requirements, and the varying productivity of software developers, has a significant impact on cost estimating and cost management.

The *PMBOK® Guide* states that the ability to influence costs is greatest at the early stages of the project, making early scope definition or time-boxing of the project critical to estimating costs. Stable architecture, testing capabilities, and enabling technologies (such as configuration management, testing, and deployment tools) have a strong influence on software cost—especially the cost of late changes. Flexible or scalable architecture, continuous testing, and enabling technologies can also reduce the long-term cost of using, maintaining, and supporting a software product.

The financial benefit of conducting the project can be continuously evaluated during evolution of the product. Each tradeoff in scope and implementation details is based on predicting and analyzing the prospective financial performance of the project's product. As the project progresses, and the understanding of the product progresses, adaptive projects will have varying degrees of emerging scope. Scope management for software projects is addressed in Section 5 of this software extension.

Figure 7-1 provides an overview of Software Project Cost Management; it is an adaptation of Figure 7-1 in the *PMBOK® Guide*.

4136


7.1 Plan Software Project Cost Management
.1 Inputs
.1 Project management plan
.2 Project charter
.3 Enterprise environmental factors
.4 Organizational process assets
.5 Software project charter
.6 Direct cost drivers
.7 Governance policies
.8 Portfolios
2. Tools & Techniques
.1 Expert judgment
.2 Analytical techniques
.3 Meetings
.3 Outputs
.1 Cost management plan
.2 Level of estimate
.3 Units of measure
.4 Cost performance measurement method
Software Project Cost Management Overview
7.2 Estimate Software Project Costs
.1 Inputs
.1 Cost management plan
.2 Human resource management plan
.3 Scope baseline
.4 Project schedule
.5 Risk register
.6 Enterprise environmental factors
.7 Organizational process assets
.8 Software size and complexity
.9 Rate of work for software projects
2. Tools & Techniques
.1 Expert judgment
.2 Analogous estimating
.3 Parametric estimating
.4 Bottom-up estimating
.5 Three-point estimating
.6 Reserve analysis
.7 Cost of quality
.8 Project management software
.9 Vendor bid analysis
.10 Group decision-making techniques
.11 Time boxed estimating
.12 Function point and source lines of code estimating
.13 Adaptive story point estimating
.14 Estimating reusable code effort
.15 Price to win for software projects
.3 Outputs
.1 Activity cost estimates
.2 Basis of estimates
.3 Project documents updates
7.3 Determine Software Project Budget
.1 Inputs
.1 Cost management plan
.2 Scope baseline
.3 Activity cost estimates
.4 Basis of estimates
.5 Project schedule
.6 Resource calendars
.7 Risk register
.8 Agreements
.9 Organizational process assets
.2 Tools & Techniques
.1 Cost aggregation
.2 Reserve analysis
.3 Expert judgment
.4 Historical relationships
.5 Funding limit reconciliation
.3 Outputs
.1 Cost baseline
.2 Project funding requirements
.3 Project documents updates
7.4 Control Software Project Costs
.1 Inputs
.1 Project management plan
.2 Project funding requirements
.3 Work performance data
.4 Organizational process assets
.2 Tools & Techniques
.1 Earned value management
.2 Forecasting
.3 To-complete performance index (TCPI)
.4 Performance reviews
.5 Project management software
.6 Reserve analysis
.7 Management metrics
.3 Outputs
.1 Work performance information
.2 Cost forecasts
.3 Change requests
.4 Project management plan updates
.5 Project documents updates
.6 Organizational process assets updates

4137

Figure 7-1. Software Project Cost Management Overview

7.1 Plan Software Project Cost Management

As stated in the *PMBOK® Guide*, Plan Cost Management is a process that establishes the policies, processes, and documentation for planning, managing, executing, and controlling project costs. This includes identifying incremental funding models and establishing change control to manage variations from the cost control plan. The inputs, tools and techniques, and outputs for planning cost management in Section 7.1 of the *PMBOK® Guide* are applicable to planning cost management for software projects, with the following additions and extensions:

7.1.1 Plan Software Project Cost Management: Inputs

The inputs in Section 7.1.1 of the *PMBOK® Guide* are applicable for planning cost management for software projects, with the modification of 7.1.1.4 and the addition of the inputs in 7.1.1.5 through 7.1.1.8.

7.1.1.1 Project Management Plan

See Section 7.1.1.1 of the *PMBOK® Guide*.

7.1.1.2 Project Charter

See Section 7.1.1.2 of the *PMBOK® Guide*.

7.1.1.3 Enterprise Environmental Factors

See Section 7.1.1.3 of the *PMBOK® Guide*.

7.1.1.4 Organizational Process Assets

In addition to those described in 7.1.1.4 of the *PMBOK® Guide*, organizational process assets for software project cost management include governance documents and the product portfolio.

7.1.1.5 Software Project Charter

In addition to providing the summary budget from which the detailed project costs are developed, the project charter also highlights the overall goals of the project and the project approval requirements that will influence the management of the project costs. These will form the focus for the tradeoff decisions that will be made throughout the project.

7.1.1.6 Direct Cost Drivers

Estimated size and complexity of software are highly correlated with effort for software projects, and therefore drive software cost; large and more complex products require more effort. Other cost drivers include skills and abilities of software developers; interactions among develop team members; relationships with customers and other stakeholders; and organizational influences. Historical values of these cost drivers and their impact on effort (i.e., cost) are often maintained as an organizational asset for various domains for which the organization develops software.

7.1.1.7 Governance Policies

In some organizations, top-level executive governance controls the objectives, policies, and procedures that may have a significant impact on the cost management plans for software projects. The organizational governance may impose a standard set of estimating, cost reporting, or software testing or review processes which need to be included in project cost. Particularly for software with safety, security, health, or financial impact for the users, policies or regulations may impact the governance controls to be built into the software. These may lead to more complex software that checks to ensure processes are being computed properly (checks and balances in intermediate results, user intervention in completing or safely terminating a software process), limits access to authorized groups of people who perform some functions (segregation of duties), or preserves audit records of who performed certain functions (such as adjusting a paycheck).

Operating policies and procedures, and the resulting software functions and controls, may be based on IT governance standards and guidance from sources such as COBIT (Control Objectives in Information and related Technology), COSO (Committee of Sponsoring Organizations of the Treadway Commission), ITIL® (Information Technology Infrastructure Library), CMMI® (Capability Maturity Model Integration), ISO/IEC 20000 (IT Service Management) or ISO/IEC 27000 (Systems and Software Security Engineering). Other inputs to software project cost include fiduciary requirements or government regulations that mandate the financial and security controls to be built into the software system. For example, the Sarbanes Oxley Act (SOX Compliance), Basel-III, Health Insurance Portability and Accountability Act (HIPAA), and others may need to be included in the software project cost management purview.

7.1.1.8 Portfolios

Priorities and constraints on an organizational portfolio of software and projects may provide inputs to planning cost management for a software project. The availability of COTS/Open Source software influences how much of the desired software will have to be original development or new integration/service development work. Even when COTS sources are available, an organization may decide to add to its portfolio and build new software, developing wholly owned Intellectual Property (IP) for future reuse or resale.

7.1.2 Plan Software Project Cost Management: Tools and Techniques

The tools and techniques in Section 7.1.2 of the *PMBOK® Guide* are also applicable to planning cost management for software projects, with the modifications to 7.1.2.2 and 7.1.2.3.

7.1.2.1 Expert Judgment

See 7.1.2.1 in the PMBOK® Guide.

7.1.2.2. Analytical Techniques

Organizations apply historical data and policy to determine the financial control limits

and decision thresholds for software projects. This is especially useful for adaptive life cycle projects where performance data is collected for each cycle of software development. When control limits are set too low, there can be significant overhead and delay introduced through revisions that require change control. An unacceptable level of risk is introduced when control limits are set too broadly.

7.1.2.3. Meetings

After the preliminary cost management plan is drafted and proposed control limits are established, a meeting is typically held with the project sponsors to reach agreement on the cost management plan.

7.1.3 Plan Software Project Cost Management: Outputs

The output in Section 7.1.3 of the *PMBOK® Guide* is also applicable to planning cost management for software projects, with the following modification. In addition, the outputs in 7.1.3.2 – 7.1.3.4 also apply to planning software project cost management.

7.1.3.1 Cost Management Plan

The cost management plan for a software project typically includes the level of estimate, units of measure and the cost performance measurement method. The values of these attributes can differ significantly in for software projects.

7.1.3.2 Level of Estimate

The level of estimate accounts for the accuracy of a cost estimate, as compared to actual cost. In general, software estimation is error-prone, and predicting the accuracy of an estimate is difficult. A rough order-of-magnitude estimate is typically generated at the initiation stage of a software project, when requirements are immature, the actual parameters of the development are being formulated, and the development team may or may not have been identified. Estimation accuracy can deviate by as much as ±150% or more at this point. A budgetary estimate might be created when the requirements or feature set and high-level design is stable or the project team and schedule have been set. This estimate may deviate by ±50%, depending on the complexity of the design, the stability of the requirements, and the skills of the team that will develop the software. A definitive estimate for a development cycle of 2-4 weeks might be accurate to within ±10% of actual cost; however, that depends on factors such as the stability of the design and accurate translation of the customer's story into product requirements.

Inaccurate estimates made to a high level of precision may not be worth the time and effort it takes to develop them. Early, rough estimates are more likely to be cost-effective, provided they are refined as the project evolves and uncertainties are resolved.

7.1.3.3 Units of Measure

The cost management plan for a software project typically includes a definitive unit of measure for each project metric such as person-hours for effort measurement, and a size measure such as Source Lines of Code (SLoC) or Function Points (FPs). More holistic measures of software such as test cases, use cases or user stories are also used. Note that a large amount of SLoC does not necessarily correspond to higher business value of the software or to the completion of required software functions in usable software. For each unit of measure like SLOC and Function Point (FP) it is necessary to introduce a concept to quantify (e.g., size of function point or SLOC) and a base unit (e.g., data

movement or statement) that helps to obtain the base quantity (e.g., number of data movements or number of statements).

7.1.3.4 Cost Performance Measurement Method

Methods of performance measurement are specified in the cost management plan. Adaptive life cycle software projects, for example, use performance trends based on estimated amount of work needed versus the actual work performed to develop working, deliverable software. This can be reflected in measures such as velocity or earned value reporting for adaptive life cycle projects and shown in visual presentations such as burn-down charts and continuous flow diagrams.

7.2 Estimate Software Project Costs

The inputs, tools and techniques, and outputs for estimating project costs in Section 7.2 of the *PMBOK® Guide* are applicable to estimating costs of software projects, with the following clarifications and extensions.

Software project managers tend to use multiple estimation approaches and then reconcile the differences among the estimates. Estimates of software project cost may need to include a number of additional factors beyond development and deployment costs, such as licensing fees for vendor software included in the software product and infrastructure upgrades for internal systems. Some of these costs may be captured in corporate overhead, such as the infrastructure resources and tools for development operations. On other projects, infrastructure resources and tools may be seen as direct charges to the software project, or assessed on a per seat basis for the team.

- 0**Project direct costs.** The dynamics of individual performances, team skills, complexity of the software product, and integration with other systems are critical success factors for software projects. Because software development projects are effort-intensive, the primary cost for most software projects is the cost associated with the people who develop the software. Productivity, skills, and motivation are widely variable among software developers, so effort data from previous projects may not be directly comparable.
- 0Fiduciary requirements and government regulations. Meeting statutory or regulatory constraints may need to be included in a software project cost estimate.
- 0Standards compliance. Some software projects may include costs for conforming to standards that are part of the organizational governance framework. However, conformance to process standards is considered to reduce project risk and cost of rework, resulting in lower overall life-cycle costs.
- 0Organizational changes. The cost of organizational changes that may impact the actual cost of a software project is typically included in the cost estimate.
- 0Cost and value risk. For some software projects, the probability that the product may not return the value anticipated can impact the cost planning of the project or lead to incremental estimates at milestones when the project is re-evaluated.
- 0Funding costs. Additional funding cost factors may include TOC (total ownership cost), payback period, break-even point, and return on investment. For iterative-incremental and adaptive life cycle projects, it may be possible to deliver working software early in the development life cycle. This can provide early payback to the sponsoring organization. The impact of the time-value of money can be reflected in the business case. Benefits that are to be realized from the project or from project's product are often considered only after the fact.

7.2.1 Estimate Software Project Costs: Inputs

The inputs for estimating project costs in Section 7.2.1 of the *PMBOK® Guide* are applicable to estimating costs of software projects, with the modification of 7.2.1.3 –

7.2.1.5. In addition, the inputs in 7.2.1.8 and 7.2.1.9 apply.

7.2.1.1 Cost Management Plan

See Section 7.2.1.1 in the PMBOK® Guide.

7.2.1.2 Human Resource Plan

See Section 7.2.1.2 in the PMBOK® Guide.

7.2.1.3 Scope Baseline

Theoretically, the fixed scope and requirements set of a predictive project can result in a reliable initial cost estimate. In contrast, many successful large software projects use feature driven delivery (FDD) where high-level scope and a set of candidate features, use cases or “epics” (complex, overarching user stories) are defined early in the project and are then evolved as uncertainties are resolved. Using an adaptive approach for a software project intentionally limits upfront planning to high-level scope, which in itself may not be a sufficient basis for an accurate initial cost estimate.

7.2.1.4 Project Schedule

As described in Section 6 of this software extension, adaptive software project life cycles make minimal initial plans and evolve the plans, including the project schedule, as the project evolves. The schedule versus priority of features to be implemented is continually evolved.

7.2.1.5 Risk Register

As described in Section 11 of this software extension, all projects (predictive, iterative-incremental, or adaptive) can benefit from initial risk analysis. Confidence in a cost estimate is dependent on the probability of and the potential impact of identified risk factors, such as the availability of subject matter experts (SMEs). Opportunity management is also pursued to identify opportunities for cost savings and additional investment/benefit returns. Risk analysis is particularly important in estimating the cost and price to be bid for a competitively sourced software project.

7.2.1.6 Enterprise Environmental Factors

The level and maturity of the product architecture, whether for a single product or the entire enterprise, has a significant impact on the effort and thus the cost of software development. Use of an existing architecture often lowers the amount of effort required for design, while it imposes constraints on the solution, particularly in the use of COTS or other non-developmental items. Once architectural decisions are made, some development tasks can be performed in parallel, thus allowing shorter schedules at higher completion rates. The result for the cost estimate is a reduced time span when the LOE costs of the supporting processes such as the PM, CM, and QA are needed.

7.2.1.7 Organizational Process Assets

See Section 7.2.1.7 in the *PMBOK® Guide*.

7.2.1.8 Software Size and Complexity

Software size and complexity are the most important factors that affect software cost, so size is a key input to most cost models. Deriving an appropriate size estimate is neither straightforward nor trivial, due to uncertainty of the requirements. Even during late stages of software development, the process of sizing is subject to wide variations. Sizing techniques include analogy, expert judgment (including Delphi), historical data, rules of thumb, and estimation algorithms (calibrated using local historical data). Because of the uncertainties associated with size estimation, estimators typically use more than one approach to estimating size. Sizing estimates are typically revised periodically as the overall cost estimate is updated.

Often, software estimates are made on small parts and rolled up (bottom-up estimation). If estimates are made only on the work to be performed to complete the software components, the cost of integration and testing of the integrated components is added. Continuous integration and testing is a key aspect of adaptive software projects.

7.2.1.9 Rate of Work for Software Projects

Organizations with stable cross-functional teams that have worked together over time can establish a predictable rate for producing working, deliverable software. This rate is called velocity; it can be used to provide accurate estimates across the full scope of software development.

7.2.2 Estimate Costs: Tools and Techniques

After determining the project and product scope, and planning software project cost management, the software project manager and project team can estimate the cost to deliver the product. The first level of estimation is typically a preliminary high-level estimate based on features to be implemented, which is used to drive initial planning. The goal of initial estimation is to quickly converge on a reasonable high-level estimate. Analogies, historical data, and expert judgment are typically used at this point.

Experts may be asked, either individually or as a group (perhaps using a Delphi process), to develop estimates. Since each expert may use a different estimation method, some perspective on the realism of individual estimates is provided. This approach may be time consuming, and is only as good as the experts' judgment. It can be especially useful when a software project involves new technologies.

- 0**Estimation Units**. The units of measure adopted by a project team or a software organization used to estimate project work may be in work units or ideal time. Both approaches exclude non-programming time and non-development time.
- 0**Work Units**. A work unit is a relative measure that is not treated as actual effort and time. Implementation of function points, for example, can be used to determine the relative amount of work required to implement a software feature, compared to implementing function points for other similar features. After a team has worked through several iterative cycles together and achieved a consistent velocity, their work units can be more accurately aligned to units of actual time and effort.

Some adaptive methods utilize *story points* or use case points for estimating. A story point is an approximation of the complexity of software functions, as expressed in a narrative of user interactions with the system (the user story). Story points are comparisons of the complexity of a new story to a well-defined base story commonly understood by the team. Story points are then awarded from a specific range of values in comparison to the base story. Some teams use a story point range defined as a modified Fibonacci sequence [i.e., 1, 2, 3, 5, 8, 13, 20, 40, 100] to scale the complexity of stories. If the base story represents 5 story points, then a 3-point story would take 60% of the base story's work to complete. Note that these are relative rather than absolute values, and may differ from team to team and project to project, limiting their

applicability across an organization or generally.

- 0**Ideal Time**. The time expected for an "ideal" software developer or a development team to deliver a feature or complete another task, without regard to actual time used for distractions, overhead functions, lost time such as holidays or to recover from disasters such as lost code.

The tools and techniques for estimating project costs in Section 7.2.2 of the *PMBOK® Guide* are applicable to estimating costs of software projects. They are expert judgment, analogous estimating, parametric estimating bottom-up estimating, three-point estimating, reserve analysis, cost of quality, project management software, vendor bid analysis, and group-decision making techniques. The following modifications and additions (7.2.2.11 – 7.2.2.15) also apply.

7.2.2.1 Expert Judgment

See Section 7.2.2.1 of the *PMBOK® Guide*.

7.2.2.2 Analogous Estimating for Software Projects

A software project team that has worked together to develop software in the past can effectively estimate the number of work units they can deliver in a given amount of time.

Some algorithmic approaches utilize analogous historical values of productivity to estimate future projects. Early estimates are often based on nominal measures such as small, medium, and large.

Software development teams, when using an adaptive approach, develop the ability to estimate their velocity more accurately. A team's velocity (amount delivered over a given period of time) can be used to estimate future effort. Velocity becomes a more accurate predictor after a team has completed several iterations together; it may not be applicable for a team that has not worked together until some performance data on the current project is collected.

7.2.2.3 Parametric Estimation Models for Software Projects

Software project managers use most of the estimation models listed in the *PMBOK® Guide*, but different approaches are used in different situations. Parametric estimation models are used to make estimates for large-phased projects, where the law of large numbers can compensate for variation in individual productivity. Analogy and expert judgment are used as for most software projects, particularly for small, highly adaptive life cycle projects.

Widely used software estimation models typically include a primary estimation algorithm with parametric adjustment factors for specific cost drivers derived from historical data. These models are calibrated for the specific software development organization, infrastructure tools, complexity of the software to be developed, and experience or ability of the team personnel, in order to produce reasonably accurate estimates. Most estimation models for software projects use some measure of product size as the primary input variable, such as source lines of code or function points. Some models permit the estimator to use raw data on the size of the project, effort, and cost drivers to calibrate the own model. Some models include functions to adjust the effort based on the size of the projects and cost drivers. COCOMO II is an example of a software estimating model. There is little established common data from industry with which to calibrate a predictive or parametric model, particularly since the infrastructure tools and methods used on projects are frequently changed for newer technology, and the historical productivity data is often closely held by an organization.

7.2.2.4 Bottom-Up Estimating for Software Projects

Bottom-up estimation is often used to estimate effort and cost of software projects. The large number of variables that can impact an estimate requires making many assumptions that need to be documented (and tracked in the risk register as well). It is also important to include, in the cost estimate, the effort to integrate and test the software components.

7.2.2.5 Three-Point Estimating for Software Projects

Estimating software code size can be based on expert judgment and the three-point PERT algorithm. Experts estimate code size as SL, (the smallest size); and SM (the 50% probable size), and SH (the largest likely size) for each software component. Lowest likely and largest likely sizes are stated at certain levels of probability (typically 5% and 95% or 20% and 80% probable). The PERT algorithms are used to estimate the mean and standard deviation of size ranges for each software component and the mean and standard deviation for the collection of components. These probability distributions can be used to calculate effort at various level of probable size.

7.2.2.6 Reserve Analysis for Software Projects

PERT three-point estimates, analogy estimates, and algorithmic estimates may be used to establish the reserve, at a given level of probability, to be included in the project estimate and the project budget. Historical project artifacts can also be analyzed to support the amount of reserve to be included for a new project by determining the difference between the known effort at the start of the last project and the amount of effort that was eventually required to deliver the project.

7.2.2.7 Cost of Quality for Software Projects

Cost of quality and the cost of other nonfunctional requirements exert a significant impact on software cost estimation. High quality (such as safety-critical or mission-critical software) can multiply the effort and thus the cost of development. Identifying quality-critical features and function up front can reduce the overall cost, as opposed to testing them in at the end of a software project. Failure Modes and Effects and Criticality Analysis (FMECA) and the processes in RTCA DO-178B/C for safety-critical avionics software are systems engineering tools that support identification of quality-critical cost factors. Also, results chains and business process analyses can identify high cost but possibly low-value, nonfunctional requirements. Such nonfunctional requirements can be expensive to implement and undetectable by the user in the operating environment.

At the same time, failing to estimate and include the cost of resources needed to meet legitimate requirements for performance, security, safety and other nonfunctional requirements can inhibit market or customer acceptance and cause huge additional costs in rework at the end of a project when the rework is most expensive. The cost impact of making software readily usable by humans for real-time, critical functions (and demonstrating its usability) is another aspect of the cost of quality.

Cost of quality also includes the cost to fix functional or technical defects found in the course of the project. These costs include effort associated with reestablishing the development environment to verify fixes, interrupting the flow of value-added work, and updating project artifacts related to the defective code. These costs of rework can be considerable and are very difficult to anticipate at the beginning of a project.

For adaptive life cycle software projects with stable teams and a history of delivery, a

historical velocity (analogy estimating) may be used to account for much of the cost of quality. In other projects, expert judgment, estimating models, and reserve analysis from prior projects can be used to establish a management reserve to handle the uncertainty associated with the cost of quality.

A key to reducing these costs in adaptive life cycle projects is gathering feedback early in the process. See also Section 8 of *the PMBOK® Guide* for more information on the cost of quality for software projects.

7.2.2.8 Project Management Software

See Section 7.2.2.8 of the *PMBOK® Guide*.

7.2.2.9 Vendor Bid Analysis

See Section 7.2.2.9 of the *PMBOK® Guide*.

7.2.2.10 Group Decision-Making Techniques

See Section 7.2.2.10 of the PMBOK® Guide.

7.2.2.11 Time-Boxed Estimating for Software Projects

Adaptive projects that are time-boxed with an evolving product scope take care that their cost estimates are not just Level of Effort (LOE) aggregates. The resources available and the current production rate determine cost. For example, if a backlog of software features is required to be delivered in 12 months and 5 people are available, then the effort will be 60 person-months. Although this approach sometimes produces an accurate estimate, care must be taken because it may provide unrealistic estimates unless the requirements and features to be included are scaled to what can be done by those 5 people in 12 months.

7.2.2.12 Function Point and Source Lines of Code Estimating

Historically, the estimated number of source lines of code [SLOC] or function points is used as the primary input variable for effort estimation. Function point estimates are considered more accurate and more easily applied from one project to another, since SLOC vary significantly by programming language and by programmer for the same function. ISO/IEC 20926, *Software and systems engineering—Software measurement—IFPUG functional size measurement method 2009*, provides a guide for software size estimation.

7.2.2.13 Adaptive Story Point Estimating

The use of adaptive life cycles for software projects avoids a complete upfront requirements or product design decomposition to make a baseline estimate. These projects intentionally use different sizing units than are traditionally used to estimate software size. The values assigned to the estimation units are specific to each software project and are not generally transferable to other projects. The values are used in the context of a project and are not compared to similarly named units in other projects.

7.2.2.14 Estimating Reusable Code Effort

Software project estimators take into account whether software code will be developed from scratch, will be reused as is, adapted from a previous project, or some combination thereof. The amount of effort required to reuse code without modification may be small.

Integration testing to check that the reused code was integrated correctly may be all that is required. In some cases, additional effort may be required to modify the existing code base to accommodate the reused code. Adapted code requires some amount of redesign, recoding, and testing. The amount of effort required depends on the amount of modification required. It is possible that (1) the adapted code may have the correct design but requires conversion because the new software is in a different language; or (2) the adapted code may require some amount of redesign to change or add capabilities. Some estimation models include parameters to account for estimated effort of reuse.

7.2.2.15 Price-to-Win for Software Projects

A further step from estimating the cost of performing a software project is estimating the price. Especially in competitive acquisitions, price is cost plus profit or fee, that is, what the customer pays. The price-to-win needs to be a price the customer is willing to pay (not exceeding the customer's budget for the project), and slightly lower than other competitors are expected to bid, but not so low that it will be rejected by the customer's evaluators as unreasonable or showing that the supplier does not understand the project. A risk evaluation is performed on the price to win (as described further in Section 11) so that the risk of having to perform the project at that price is acceptable to the supplier's organization. (This discussion ignores the competitive strategy of bidding a small project at a price with a low probability of being profitable, or even below the most likely cost, with the intention of gaining experience or intellectual property for future, more profitably priced projects.)

7.2.3 Estimate Software Project Costs: Outputs

The outputs in Section 7.2.3 of the *PMBOK® Guide* are applicable for estimating software project costs.

7.2.3.1 Activity Cost Estimates

See Section 7.2.3.1 of the *PMBOK® Guide*.

7.2.3.2 Basis of Estimates

See Section 7.2.3.2 of the *PMBOK® Guide*.

7.2.3.3 Project Documents Updates

See Section 7.2.3.3 of the *PMBOK® Guide*.

7.3 Determine Software Project Budget

The inputs, tools and techniques, and outputs for determining budget in Section 7.3 of the *PMBOK® Guide* are applicable to determining the budget for software projects.

7.3.1 Determine Software Project Budget: Inputs

The inputs for determining budget in Section 7.3.1 of the *PMBOK® Guide* are applicable to determining the budget for a software project.

7.3.1.1 Cost Management Plan

See Section 7.3.1.1 of the *PMBOK® Guide*.

7.3.1.2 Scope Baseline

See Section 7.3.1.2 of the *PMBOK® Guide*.

7.3.1.3 Activity Cost Estimates

See Section 7.3.1.3 of the *PMBOK® Guide*.

7.3.1.4 Basis of Estimates

See Section 7.3.1.4 of the *PMBOK® Guide*.

7.3.1.5 Project Schedule

See Section 7.3.1.5 of the *PMBOK® Guide*.

7.3.1.6 Resource Calendars

See Section 7.3.1.6 of the *PMBOK® Guide*.

7.3.1.7 Risk Register

See Section 7.3.1.7 of the *PMBOK® Guide*.

7.3.1.8 Agreements

See Section 7.3.1.8 of the *PMBOK® Guide*.

7.3.1.9 Organizational Process Assets

See Section 7.3.1.9 of the *PMBOK® Guide*.

7.3.2 Determine Software Project Budget: Tools and Techniques

The tools and techniques for determining budget in Section 7.3.2 of the *PMBOK® Guide* are applicable to determining the budget for a software project, with the modification of 7.3.2.2.

7.3.2.1 Cost Aggregation

See Section 7.3.2.1 of the *PMBOK® Guide*.

7.3.2.2 Reserve Analysis for Software Projects

A software project budget is based on the sum of estimates for all identified work plus an additional reserve for work that will potentially emerge. During the project, reserved budget is either applied to meet contingencies or preserved as surplus or profit. Ideally, as the project progresses and the risks and uncertainties are resolved, the amount of reserve needed is reduced to zero by the end of a project. When charted over time, the

amount of reserve needed looks like a cone (the “cone of uncertainty,” large at the beginning of the project and narrowing to zero by the end of the project).The reserve may be divided between the amount that the project manager can use directly and the management reserve, which will require authorization to be applied to the project. Adaptive projects that adjust to changes in priorities and rolling-wave planning may find that the level of uncertainty remains higher throughout the project, affecting the need for a budgeted reserve. Conversely, after several iterations, the adaptive project team may have developed a consistent idea of velocity and lowered risk of needing the management reserve to finish the work.

7.3.2.3 Expert Judgment

See Section 7.3.2.3 of the *PMBOK® Guide*.

7.3.2.4 Historical Relationships

See Section 7.3.2.4 of the *PMBOK® Guide*.

7.3.2.5 Funding Limit Reconciliation

See Section 7.3.2.5 of the *PMBOK® Guide*.

7.3.3 Determine Software Project Budget: Outputs

The outputs for determining budget in Section 7.3.1 of the *PMBOK® Guide* (7.3.3.1 – 7.3.3.3) are applicable to determining the budget for software projects.

7.3.3.1 Cost Baseline

See Section 7.3.3.1 of the *PMBOK® Guide*.

7.3.3.2 Project Funding Requirements

See Section 7.3.3.2 of the *PMBOK® Guide*.

7.3.3.3 Project Documents Updates

See Section 7.3.3.3 of the *PMBOK® Guide*.

7.4 Control Software Project Costs

The inputs, tools and techniques, and outputs for controlling costs in Section 7.4 of the *PMBOK® Guide* are applicable to controlling costs of software projects, with the following clarifications.

Effective software project managers constantly monitor changes to stakeholder requirements and other changes as they evolve to analyze their potential impact on project cost. Some changes will be “in scope” and require no changes to effort allocations (and therefore cost) and other changes may be “out of scope” and will require changes to effort (cost) and schedule. This is especially important for adaptive life cycle software projects because stakeholder requirements are dynamic in nature, and changes can occur rapidly as the project progresses. Also, different organizations and their customers use different cost accrual methods for measuring software project success. For example, a project deliverable may be considered as value-adding after successful integration and

verification testing by some organizations and their customers, while others may consider it value-adding only after successful user acceptance testing and product acceptance.

7.4.1 Control Software Project Costs: Inputs

The inputs for controlling costs of software projects in Section 7.4.1 of the *PMBOK® Guide* are applicable to controlling costs of software projects.

7.4.1.1 Project Management Plan

See Section 7.4.1.1 of the *PMBOK® Guide*.

7.4.1.2 Project Funding Requirements

See Section 7.4.1.2 of the *PMBOK® Guide*.

7.4.1.3 Work Performance Data

See Section 7.4.1.3 of the *PMBOK® Guide*.

7.4.1.4 Organizational Process Assets

See Section 7.4.1.4 of the *PMBOK® Guide*.

7.4.2 Control Software Project Costs: Tools and Techniques

The tools and techniques for controlling costs of software projects in Section 7.4.2 of the *PMBOK® Guide* are applicable to controlling costs of software projects, with the following modification of 7.4.2.1 and 7.4.2.2, and the addition of 7.4.2.8.

7.4.2.1 Earned Value Management for Software Projects

Earned value management, as applied to projects that produce physical artifacts, and as applied to predictive life cycle software projects, is concerned with measuring cost and schedule progress against an overall plan for cost, schedule, and delivered work products. Earned value reporting can be successfully applied to predictive software projects provided the work products are tracked using control gates that give no credit for work in progress until the work product successfully satisfies its pre-specified acceptance criteria (i.e., binary tracking). Actual cost and schedule for completing the work product is then compared to planned (i.e., budgeted) cost and schedule to develop cost and schedule performance indices (CPI and SPI) that are used to project the estimate at completion (EAC), estimate to complete (ETC), and estimated completion date (ECD) using values of the current CPI and SPI. Periodic updating of CPI and SPI, based on accumulation of past performance and recent performance, provides increasingly accurate values of EAC and ECD. This approach, based on objective acceptance criteria for work products and binary control gates, can provide accurate tracking information and avoids the 95% complete syndrome that commonly occurs for predictive life cycle software projects.

For adaptive life cycle software projects, a detailed project plan is not developed upfront; the initial project plan typically provides estimates at a high level of project scope, based on estimates for implementing the identified product features. The plan is elaborated in detail as the project evolves: new features emerge and existing features are elaborated (much like rolling wave planning for predictive software projects). As software is delivered in increments of working, deliverable software, earned value management

focuses on directly measuring the performance of delivered product increments versus allocated time and effort to develop the product increments.

Adaptive development of software evolves in an iterative, nonlinear fashion, with feedback loops that affect the initial plan. Frequent changes are to be expected throughout the project life cycle, thus measuring progress relative to the initial plan can be misleading unless the plan is frequently updated. Project management techniques for adaptive life cycle projects, such as burn charts that indicate the amount of functionality delivered over time versus the amount of functionality remaining to be developed, provide status and progress information very similar to earned value measurement for predictive life cycle projects. Costs can be added to the charts to view cost performance together with rate of feature completion.

For predictive life cycle projects, deviations from the budgeted actual cost and scheduled completion date may indicate the need to: (1) apply contingency reserve funds and reserved schedule time to bring project performance into alignment with the project plan; (2) modify the plan to reflect actual performance; or (3) use a combination of updating the plan and applying contingency reserves. For adaptive projects, additional effort and schedule may be allocated to a feature when detailed planning shows more time and effort is needed than was estimated, provided the allocations are within the bounds of control accounting. Change control is exercised to access management reserve when the control reserve is exhausted.

7.4.2.2 Forecasting for Software Projects

Desirable attributes of a software development forecasting method include providing credible estimates in a short amount of time, quickly communicating the need for decisions or actions, and empowering the project sponsors to choose how their software development funds are to be spent. Earned value tracking, burn-down charts, and cumulative flow diagrams (CFDs) provide indicators of the costs expended to-date on a project and provide forecasts of project cost at completion. Earned value charts, burn-down charts, and cumulative flow diagrams typically report cost in units of labor (i.e., staff-hours) or in monetary units that account for labor costs or may include additional costs.

The information is presented as calculated amounts, but it is the visibility of the charts that is most valuable to project managers, software teams, and other stakeholders. The charts indicate cumulative progress, how much effort or money is being expended on the project, and how much remains to be done. This is important because it represents the amount of effort or money needed to keep the project operational regardless of the amount of work assigned to be done. A simple calculation of the resources that are needed, the percentage of allocation, and all costs associated can be placed on the earned value, burn-down or cumulative flow chart. Often there is a major effort to re-estimate and re-baseline a large project and significant customer discussions may occur concerning scope adjustment and the deferral and prioritization of product features. On smaller projects, it may be simpler to do an extrapolation of how the cost and schedule need to be adjusted in order to deliver the desired software functionality and to then adjust the scope to stay within budget. Cost is controlled as an independent variable. It is up to the economic decision maker to adjust the functionality to be developed so that it can be completed within the specified budget and time, or to adjust the budget and time, or some combination thereof.

7.4.2.3 To-Complete Performance Index (TCPI)

See Section 7.4.2.3 of the *PMBOK® Guide*.

7.4.2.4 Performance Reviews

See Section 7.4.2.4 of the *PMBOK® Guide*.

7.4.2.5 Project Management Software

See Section 7.4.2.5 of the *PMBOK® Guide*.

7.4.2.6 Reserve Analysis

See Section 7.4.2.6 of the *PMBOK® Guide*.

7.4.2.7 Management Metrics for Software Projects

Earned value graphs, burn-down charts, and cumulative flow diagrams that are based on planned versus actual cost, time, and product features depict the software cost measurement for project control.

- 0**Earned value graphs.** A traditional earned value graph for a predictive life cycle project displays budgeted and actual cost on the vertical axis versus time on the horizontal axis. Cumulative trend lines based on periodic earned value reports display the deviations of planned versus actual cost and planned versus schedule progress as well as the projections of estimated actual cost and estimated completion date versus the estimated actual cost and estimated completion date.
- 0**Burn-down charts**. A burn-down chart is a graphical representation of remaining work versus time. The remaining work (i.e., backlog) is typically displayed on the vertical axis, with time along the horizontal. A burn-down chart is thus a run chart of remaining work. It can be used to visualize project completion. A set of previous burn-down charts can provide trends for the project. A typical burn-down chart is shown in Section 10.2.3.7.
- 0**Cumulative flow diagrams**. As shown in Section 6, cumulative flow diagrams (CFDs) provide a method for tracking the progress of adaptive life cycle projects. CFDs communicate absolute progress while visually providing a proportional message of total completeness because they plot both the total scope and the progress of individual activities. These diagrams can be correlated with resource expenditures to support cost control. They can also be used to track flow attributes such as work in progress and lead time for specific tasks or features.

7.4.3 Control Software Project Costs: Outputs

The outputs in Section 7.4.3 of the *PMBOK® Guide* are applicable to controlling the costs for software projects..

7.4.3.1 Work Performance Information

See Section 7.4.3.1 of the *PMBOK® Guide*.

7.4.3.2 Cost Forecasts

See Section 7.4.3.2 of the *PMBOK® Guide*.

7.4.3.3 Change Requests

See Section 7.4.3.3 of the *PMBOK® Guide*.

7.4.3.4 Project Management Plan Updates

See Section 7.4.3.4 of the *PMBOK® Guide*.

7.4.3.5 Project Documents Updates

See Section 7.4.3.5 of the *PMBOK® Guide*.

7.4.3.6 Organizational Process Assets Updates

See Section 7.4.3.6 of the *PMBOK® Guide*.

8 Software Project Quality Management

According to the *PMBOK® Guide*, Project Quality Management includes the processes and activities of the performing organization that determine quality policies, objectives, and responsibilities so that the project will satisfy the needs for which it was undertaken. Project Quality Management involves three main processes: Plan Quality, Perform Quality Assurance, and Perform Quality Control. Figure 8-1 provides an overview of software project quality management. Most of the tools and techniques for project quality management in Section 8 of the *PMBOK® Guide* are applicable to software projects. This section presents considerations that are unique to, or especially important when planning software quality management, performing software quality assurance, and controlling software quality. This section also discusses how software quality is defined and how software project managers plan for, perform, and control software quality management. This section covers both product quality and process quality; that is, how software managers and teams plan for, perform, and control the processes to ensure the quality of the delivered software product, and how to improve the quality of processes used on software projects and in software organizations.

Software quality is relative, and not an absolute to be attained. The *PMBOK® Guide* defines quality as delivered performance: "the degree to which a set of inherent characteristics fulfill requirements." The definition from software engineering is similar: "the degree to which a software product satisfies stated and implied needs when used under specified conditions." **[21]**. To paraphrase Drucker: "Quality of a software product or service is not what the developers put in. It is what the customer gets out and is willing to pay for" **[22]**. For example, a software product used for checking the whereabouts of one's friends may not need the same qualities of timeliness, accuracy, and precision as software used to guide missile trajectories. Attainment of high levels of quality attributes is more prevalent in critical software, which has impact on human health, safety, and welfare, and for the protection of personal or business proprietary information.

Quality reflects the requirements or needs, which may initially be unstated. As previously discussed in Section 5, skilled software engineers elicit needs from the software users and other stakeholders. ISO/IEC/IEEE Std 830 provides an extensive list of the types of requirements that should be considered but sometimes are not. Nevertheless the project needs to generate an acceptable level of quality in fulfilling those requirements.

The *PMBOK® Guide* states that performing quality assurance is the process of auditing the quality requirements and the results from quality control measurements to ensure that appropriate quality standards and operational definitions are used; and that quality control is the process of monitoring and recording results of executing quality activities to assess performance and recommend necessary changes. See also **[31]**.

Software Quality Assurance (SQA) is primarily concerned with assessing and improving the quality of the processes used to develop software. Because software is developed by individuals and teams who use processes and apply tools and techniques, it is a

fundamental principle in software engineering that high-quality processes, when executed by competent personnel, result in high-quality products. *Software Quality Control* (SQC) is primarily concerned with the methods, tools, and techniques applied to the software work products (including but not limited to software code) to ensure that the work products will satisfy the quality requirements in a cost-effective manner.

Successful software development organizations follow consistent and repeatable processes and focus on improving those processes over time. This approach results in improvements in the resulting software products and advantages for the producing organization. These improving processes are not rigidly segmented steps, but allow room for learning and adjustment to situation-specific judgment during software development. Mature teams find a balance between process rigor and situation-specific decision-making. SQA and SQC play critical roles in helping the development organization tune their processes to maximize performance and achieve the potential for project success.

In organizations that use predictive approaches to software development, software quality assurance (SQA) and software quality control (SQC) have prominent roles near the end of the life cycle, using an independent interpretation of the requirements as the control. This deferred involvement of QA can lead to great pressure on project management, eager to release the software and complete the project, to ignore or suppress quality findings of software defects and to shortened or inadequate software testing.

For reasons of objectivity in evaluating quality, the *PMBOK® Guide* recommends independence of quality audits. Independence can be obtained by using a third-party organization for quality control, setting up a different line of reporting for the QC person on the project, or at the least, having a different developer perform peer review or unit tests.

Mature teams work together in collaboration to achieve quality, rather than viewing QC as an adversary. In smaller projects, SQA personnel may even be members of the development team. While larger organizations may mandate a separation of SQA and SQC personnel from development activities, so that SQA and SQC can investigate, audit, and recommend changes based on newly recognized opportunities or issues that arise, collaborative exploration is still easily achieved within cross-functional product teams. On the other hand, when SQA and SQC are independent functional groups, and are not included in cross-functional teams, collaborative exploration of quality issues may be lost and SQA and SQC then become adversaries to software development.

The user role in determining whether software has acceptable quality may be played by different persons, depending on the project's context. In a commercial software product company, the user may be represented by the product manager or documented in a fictional persona that provides the knowledge, tasks, and interests of a typical user. In an IT enterprise project, the user may be an authorized subject matter expert in the business processes the software will serve. For a software project conducted under contract, the accepting authority of the acquiring organization may represent the user. It is important for software project managers to ensure that the project team understands that the *user* is the person or persons who defines what "quality" and "fit for use" mean in order to satisfy user needs. However, the project manager must always keep in mind that the users may not be able a priori to enunciate what they really want and need. For this reason, the Project Manager must rely on the expertise of Business Analysts and others who know how to elicit quality expectations.

Software quality has been a fundamental issue from the early days of developing algorithms to perform mathematical calculations quickly and accurately to the present day. Beyond the basic pass/fail questions of whether the software runs and returns a usable result (sometimes called a software smoke test), the quality attributes for a particular software product may include elements from a very long list, which includes concepts that range from accessibility, adaptability, analyzability, availability, compatibility, and complexity, to survivability, testability, understandability, and usability (the well-known "ilities" of software).

Some software quality attributes are important for users (e.g., efficiency, safety, security, reliability, availability) while others are important to other stakeholders (e.g., maintainability is important to those who provide sustainment services). The ISO/IEC 25010 series of standards provides a complete list of software quality attributes aligned with different stakeholder needs.

The complexities of software quality have led to a number of quality views and models such as those in ISO/IEC 25010 **[23]** and the other standards referenced in this section. Models include process quality, product (internal and external) quality, quality in use, data quality, and quality of the software code, which is appraised by inspections or "static" testing, and by exercising the software in "dynamic" testing.

From the perspective of process quality, the manager considers: How is the work organized to produce software? Are the processes efficient and effective in achieving the project and product goals, and also in building a strong, cohesive team for ongoing work? What methods and tools are used and are they used effectively?

The internal quality model looks at the software itself as an open "white box," where reviewers can directly examine the code and the accompanying artifacts, such as design documents, even as they are being developed. Automated software tools are available to perform many aspects of white-box examination. They include static and dynamic testing tools that check for code coverage, adherence to coding standards, uninitialized variables, and many other types of coding errors.

The external quality model treats the software as a "black box": testers assess how it behaves from the input-output behavior, measure the software's performance, examine how well it performs its functions, and observe the conditions under which it breaks. External quality assessment is typically accomplished by functional black box testing, which is performed by a testing group or by a representative of the intended user community.

Even if the software passes predefined evaluations of its internal and external quality, users may still think it lacks quality. The *quality in use* perspective looks at the impact of the product on users and other stakeholders when it is used in a specific environment and context. *Usability* is defined as the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use **[24]**. Characteristics of quality in use include (1) effectiveness (gets the job done), (2) satisfaction (usefulness, trust, pleasure in use, comfort), and (3) freedom from risk (economic risk mitigation, health and safety risk mitigation, environmental risk mitigation).

A data quality model applies to how structured data is acquired, manipulated, and used by a computer system to satisfy user's needs; it includes data that can be shared within the same computer system or across different computer systems. Some examples of data quality characteristics are consistency, currency, completeness, precision, and accuracy.

Figure 8-1 provides an overview of Software Project Quality Management; it is an adaptation of Figure 8-1 in the *PMBOK® Guide*.

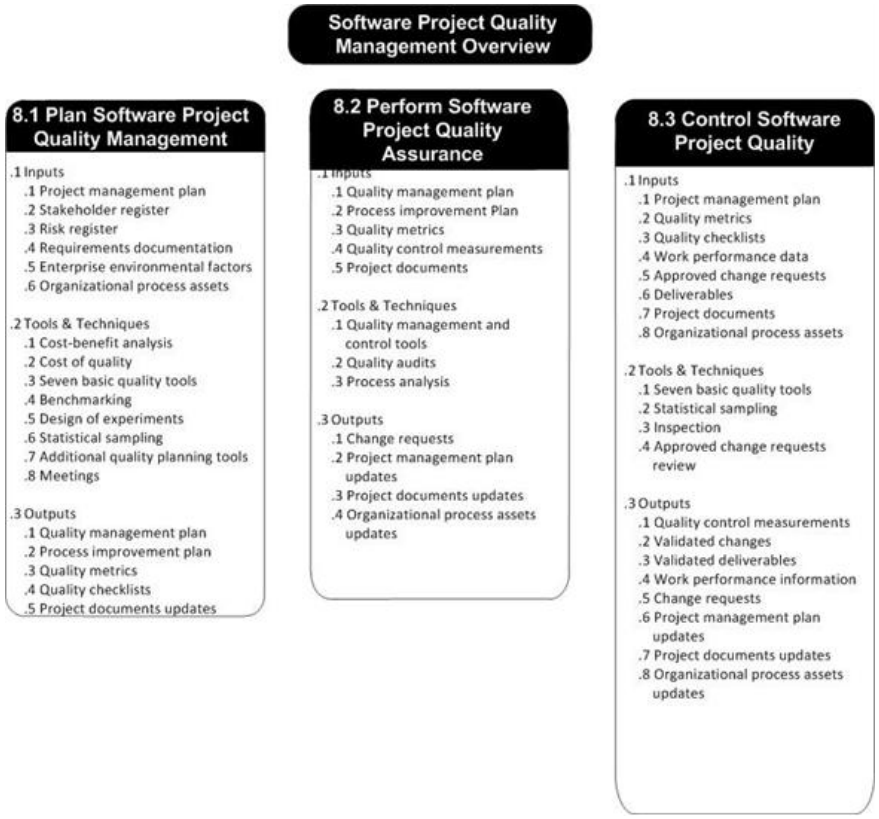


Figure 8-1. Software Project Quality Management Overview

8.1 Plan Quality Management

Most of the methods, tools, and techniques for planning quality management in Section 8.1 of the *PMBOK® Guide* are equally applicable to software projects. This section presents considerations that are unique to, or especially important when planning quality management for software projects.

Planning Software Quality Management for both process and product is an inseparable element of project planning in general. Determining the scope and goals of the project and establishing the life cycle processes to be used leads to decisions about how to integrate quality assurance and quality control into the software development processes. Identifying the criticality of the software and balancing the tradeoffs between quality, schedule, and cost determine how much emphasis will be given to product quality during the project. Defining what is acceptable quality to meet the users' needs determines when the product is ready for release and when the project can be closed. Explicit planning for process improvement can lead to midcourse changes in projects, as well as result in benefits for future projects within the organization.

A significant part of planning software project quality management activities is determining which of the software quality attributes are priorities for a particular project, and how the attributes are specified in the software requirements. Defining what quality attributes will be built into the product, and how the attributes will be measured by SQA and SQC activities, such as testing and reviews, significantly affects the scope and resources required to successfully plan and execute a software project.

ISO/IEC/IEEE Standard 15026 for Systems and software engineering – Systems and Software Assurance **[25]**, is a multi-part standard that provides a comprehensive framework for developing the appropriate assurance case(s) to guide software development projects where one or more critical properties must be achieved.

Testing provides a good example of how quality management activities span the three key processes of software quality management (planning, performing, controlling): *test*

planning is a component of Plan Quality, *defect data analysis* is a component of Perform Quality Assurance, and test *execution* is part of Perform Quality Control. But planning for SQA and SQC is more than designating a small group of testers and auditors who are budgeted proportionately to the developer team and scheduled to pick out defects at the end of a project. Since it is less expensive to "build a little, test a little" than to spend months developing and integrating a complex system that fails testing, software quality assurance and quality control need to be performed by everyone on the team, through continuing peer reviews, walkthroughs, inspections, automated regression tests, and analyses. SQA and SQC is best planned to occur as part of requirements specification, architecture and data design, and software development, as well as through configuration management and formal testing.

Adaptive software project life cycles that rely on frequent iterations to produce working, testable, and deliverable software are well suited for planning an integrated approach to SQA and SQC. For predictive software project life cycles that consist of distinct development phases, SQA and SQC are planned as distinct processes.

8.1.1 Plan Software Project Quality Management: Inputs

The inputs for planning quality management in Section 8.1.1 of the *PMBOK® Guide* are applicable for software projects. The following considerations also apply to the inputs for planning software project quality management.

In addition to the quality planning inputs identified in the *PMBOK® Guide*, software project managers typically place emphasis on identifying the stakeholders and the product requirements, as well as quality statistics from previous projects.

In general, software projects fail because the software product does not meet user expectations of functionality and quality. The software project manager is responsible for ensuring that all stakeholders understand failure to meet users' quality expectations will result in project failure. Project stakeholders can be documented in a stakeholder register. In addition to the end users and their managers, other stakeholders are those who will affect or be affected by the software product, either while it is under development or during operation after it is delivered. For example, the stakeholders of an enterprise resource planning (ERP) system include the members of IT operations who will be responsible for sustaining the ERP system; their concerns will include the interoperability, performance, and robustness of the system. Each member of the product development team is also a product stakeholder. (See Section 13 of this software extension for more considerations about communicating with stakeholders.)

Requirements for software product quality are established when the functional requirements are established. The quality requirements are an element of the requirements for product release or product acceptance. In software product companies, the quality requirements are usually included in the market requirements document (MRD). IT projects may simply use a features/backlog list. If so, the project manager needs to make sure that the quality requirements are included.

External customers may not be able to precisely state performance requirements and other nonfunctional quality requirements. A software project manager may need to engage product managers and other appropriate stakeholders in the elicitation of nonfunctional requirements to determine which quality attributes are most important to the external customers. Requirements for software product quality may also include regulatory requirements (e.g., for life-critical systems) and in contractual requirements that may be imposed on component providers and on suppliers of custom-built or customized software components.

Inputs for planning the management of software process quality typically includes quality analyses from past projects or from previous iterations of the present project. Inputs for software process quality planning are associated with various quality analyses at the

story, iteration or release (e.g., to determine whether code reviews and other types of evaluations were executed as anticipated and if they were successful); defect find/fix rates can be examined (to determine whether the numbers are rising or falling); time spent on defect fixes can be examined (to determine if they are adversely impacting planned feature development and whether reviews and testing are yielding the expected results); and previous lists of known problems and deferred defects by severity and by feature or module can be investigated (to determine whether there are error-prone areas in the software).

8.1.1.1 Project Management Plan

See Section 8.1.1.1 of the *PMBOK® Guide*.

8.1.1.2 Stakeholder Register

See Section 8.1.1.2 of the *PMBOK® Guide*.

8.1.1.3 Risk Register

See Section 8.1.1.3 of the *PMBOK® Guide*.

8.1.1.4 Requirements Documentation

See Section 8.1.1.4 of the *PMBOK® Guide*.

8.1.1.5 Enterprise Environmental Factors

See Section 8.1.1.5 of the *PMBOK® Guide*.

8.1.1.6 Organizational Process Assets

See Section 8.1.1.6 of the *PMBOK® Guide*.

8.1.2 Plan Software Project Quality Management: Tools and Techniques

The tools and techniques for planning quality management in Section 8.1.2 of the *PMBOK® Guide* are applicable for software projects. In addition to the tools and techniques listed below, the following considerations also apply to for planning software project quality management.

In addition to the quality planning tools discussed in the *PMBOK® Guide*, planning for software quality includes confirming user needs and requirements, cost-benefit analysis (CBA), cost of quality, developing a testing strategy, and selecting a defect management and quality control approach. Some comments follow:

- 0**Planning for software quality.** The users may not have experience in defining their quality expectations as testable requirements; therefore, the project team needs to be adept in eliciting the needed information. This often requires ongoing validation from the users that the software will meet their needs, using techniques such as prototypes, mock-ups, and other simulations.
- 0**Cost benefit analysis**. For most software projects, there are trade-offs among the various levels of product quality, the amount of functionality delivered, and the time and effort required to fix defects. An example of CBA is comparing the cost of testing and

rework for different levels of defect removal. Determining an acceptable level of defects may involve comparative benchmark evaluation of the relevant quality attributes in the major competitors' products. While it is natural for the team to want to correct all problems detected, the software project manager typically does not plan for a significantly higher amount of defect correction than is warranted by user expectations. For example, depending on the context of the user and user environment, there may be no need to correct a defect that will rarely be encountered and for which there is a work-around.

- 0**Cost of quality (COQ).** Analyzing and reducing the COQ for software includes costs of the following QC activities:
  - *Appraisal:* cost of finding software defects,
  - *Internal:* cost to fix defects discovered during software development or modification,
  - *External:* cost to fix software defects reported by users, and
  - *Prevention:* cost of eliminating the root causes of software defects.

Appraisal techniques for software include reviews and inspections of work products (requirements, design, code, test plans, documentation) and testing and demonstration of working software.

In software projects, the cost of quality is not just the relatively small cost of correcting the code, but the larger costs associated with the testing effort to verify the change and validate its effectiveness, to communicate the change to all affected parties, and to change the work products or processes that use the software. This cost is greatly increased once the software is in use and patches or new versions are released and applied.

Software quality planning includes developing a test policy or testing strategy. In software, the "design of experiments" is captured in the test strategy, as reflected in test plans and scenarios and the level of test coverage of the code base. Software systems may have thousands of potential branches through the code, which could be exercised with an infinite range and volume of valid and invalid inputs, running the software with previously developed modules on test hardware with more or less fidelity to the performance of the full production system, in isolation or in combination with other infrastructure software. Test planning also takes into account the need for rework, data refresh, and retest, because rarely does one cycle of testing produce completely acceptable results. Since it is almost never possible, too time-consuming, or not cost-effective to test everything, part of quality planning is choosing the test strategy, so that the most valuable and predictive tests are scheduled. Risk-based test strategy applies design, development and test resources to the areas with the most impact on successful delivery and use of the software.

A goal of planning a predictive life cycle software project is to arrange the sequence of work activities to obtain feedback from testing and reviews as early as possible. Software architects and software designers can help in identifying opportunities to build the software in a manner that provides early quality feedback.

When using adaptive software project life cycles, different levels of testing that happen at different points. Story level testing is the validation of business rules and code quality relevant to the small increment of software that has just been developed by the team. Verifying the small increments with given inputs and outputs in the same iteration cycle as development of a software increment finds defects quickly and reduces the cost of testing later in the project.

Functional testing (including feature-level testing) includes integration testing as well as quality-in-use testing across product features. Good practice is to validate the product as early as possible using sample customer databases, to mitigate the risk of regression in quality. Good practice is to coordinate work across the project so this level of testing can be performed throughout the project, rather than late in the project. If these levels of testing are being performed throughout the project, the risk of major defects being identified late in the project is reduced.

The software project manager also needs to plan for processes and systems to identify, categorize, measure, and treat defects. Defect metrics need to be defined in the project's release criteria. Release criteria are a specialized form of testable requirements and provide an input to the software release management or change management process.

Software defects are generally categorized by severity (how many users will be affected and how badly). ISO/IEC/IEEE Standard 1044 – Classification for Software Anomalies **[26]** provides guidance in establishing defect classifications that are generally meaningful for all software projects. Typically, the acceptable level of defects is specified by the planned kind of release (beta, general availability, customized). It is typical to allow none of the highest category defects to be released, but the percentage of second and third level defects often depends on the type of release and the user's expectations.

Release criteria are project specific and may involve a level of uncertainty. Defects are balanced against risk considerations (see Section 11 of this software extension). Some users prefer earlier functionality to freedom from noncritical bugs. Some projects in the safety-critical domain have very high levels of release criteria. Other software products, such as static web pages, cause very little risk to safety, but may affect reputation if poorly done. Risk management techniques are important in developing a test strategy and assessing the impact of latent defects not discovered or removed before a software release.

8.1.2.1 Cost-Benefit Analysis

See Section 8.1.2.1 of the *PMBOK® Guide*.

8.1.2.2 Cost of Quality (COQ)

See Section 8.1.2.2 of the *PMBOK® Guide*.

8.1.2.3 Seven Basic Quality Tools

See Section 8.1.2.3 of the *PMBOK® Guide*.

8.1.2.4 Benchmarking

See Section 8.1.2.4 of the *PMBOK® Guide*.

8.1.2.5 Design of Experiments

See Section 8.1.2.5 of the *PMBOK® Guide*.

8.1.2.6 Statistical Sampling

See Section 8.1.2.6 of the *PMBOK® Guide*.

8.1.2.7 Additional Quality Planning Tools

See Section 8.1.2.7 of the *PMBOK® Guide*.

8.1.2.8 Meetings

See Section 8.1.2.8 of the *PMBOK® Guide*.

8.1.3 Plan Software Project Quality Management: Outputs

The outputs for planning quality management in Section 8.1.3 of the *PMBOK® Guide* are applicable for software projects. The following considerations also apply to the tools and techniques for planning software project quality management.

In addition to the quality planning outputs discussed in the *PMBOK® Guide*, special considerations apply to the software quality management plan, quality metrics measurement plan, and quality checklists.

8.1.3.1 Quality Management Plan

See Section 8.1.3.1 of the *PMBOK® Guide*.

In addition, a software quality management plan should address configuration management topics, including: (1) version control of source code, object code, and other artifacts, and (2) control of a definitive media library for approved file versions and approved product baselines. Practices such as continuous integration, closed-loop change processes, and iteration retrospectives are typically considered. ISO/IEC/IEEE Standard 730 – Software Quality Assurance Plans **[27]** contains most of the content to be included in a quality management plan, including detailed requirements for software quality assurance plans.

8.1.3.2 Process Improvement Plan

See Section 8.1.3.2 of the *PMBOK® Guide*.

8.1.3.3 Quality Metrics

See Section 8.1.3.3 of the *PMBOK® Guide*.

A software quality management plan may also contain the quality metrics measurement plan, including such elements as software size measures, software quality measures, and acceptance thresholds. Software size measures are a way to estimate the size of the software product, which affect the size of the software quality effort. Software measures may be based on lines of code, but those can vary in different programming languages and depend on the elegance of the coding. Lengthier code that is well commented and factored into functional modules may be easier to maintain than highly compressed code, resulting in a lower overall cost of quality. Rather than measuring by lines of code, software functionality is better measured by the number of requirements, function points, number of features, use cases, user stories, and technical artifacts. Typical software quality metrics relate to the characteristic or attribute being measured and may include churn in requirements, percent of new requirements, defect find/fix ratios, and trends. Based on the nature, size, or context of the project, more metrics can be added to address specific quality attributes, such as performance, throughput, resistance to hacking, or usability of the software and documentation.

The measurement plan, quality management plan, or project management plan may also define how project efficiency and thus process quality will be measured. The most basic measurements, suitable for all types of software projects, are elapsed time and expended effort (staff-days) per function. Measures of productivity help in planning how quickly functionality can be produced, including getting found defects corrected. These indicators reflect the quality of the planning for the iteration/project, and will be input into Project Cost Management (Section 7) and Project Time Management (Section 6). These measurements are essential for projects when pursuing process improvement.

8.1.3.4 Quality Checklists

See Section 8.1.3.4 of the *PMBOK® Guide*.

Checklists are reminders to complete all steps in a procedure or test, either to train developers who are being introduced to new tools or to document that the experienced developers have not inadvertently skipped steps. In software projects, checklists cover, for example, the steps necessary to run a successful build or to check code in and out of repositories. They are one of the easiest ways to assure consistency and accuracy in performing repetitive tasks and assuring that the tasks are carried out the same way, no matter who is performing the tasks.

8.1.3.5 Project Documents Updates

See Section 8.1.3.5 of the *PMBOK® Guide*.

8.2 Perform Software Project Quality Assurance

Most of the methods, tools, and techniques for Perform Quality Assurance in Section 8.2 of the *PMBOK® Guide* are equally applicable to software projects. This section presents considerations that are unique to or especially important when planning quality management for software projects.

The *PMBOK® Guide* states that Perform Quality Assurance is the process of auditing the quality requirements and the results from quality control measurements to ensure that appropriate quality standards and operational definitions are used. In software project management, SQA involves a broader view of the entire project to ensure that processes are being performed as documented and are producing acceptable results. In this extension to the *PMBOK® Guide, software quality assurance* (SQA) is defined as a set of activities that define and assess the adequacy of the software processes used to develop and modify software products. SQA provides evidence for a justified statement of confidence that the software processes will produce software products that conform to their requirements and will satisfy users' needs.

SQA thus covers more than audits of requirements and the results of software quality measurements. SQA comprises a full set of planned and systematic activities that can be demonstrated to provide confidence that a product or service will fulfill its requirements for quality. SQA uses the classic tools of demonstration, inspection, analysis, and testing (often divided into verification testing and validation testing) as well as audits. Both SQA and SQC can involve analysis of defects and problems and recommendations for improvement.

IEEE standards that may be useful include ISO/IEC/IEEE Standard 829 – Software and System Test Documentation **[28]**; ISO/IEC/IEEE Standard 1008 – Unit Testing **[29]**; and ISO/IEC/IEEE Standard 1012 – System and Software Verification and Validation **[30]**.

8.2.1 Perform Software Project Quality Assurance: Inputs

The inputs for performing quality assurance in Section 8.2.1 of the *PMBOK® Guide* are applicable to performing quality assurance for software projects.

8.2.1.1 Quality Management Plan

See Section 8.2.1.1 of the *PMBOK® Guide*.

8.2.1.2 Process Improvement Plan

See Section 8.2.1.2 of the *PMBOK® Guide*.

8.2.1.3 Quality Metrics

See Section 8.2.1.3 of the *PMBOK® Guide*.

8.2.1.4 Quality Control Measurements

See Section 8.2.1.4 of the *PMBOK® Guide*.

8.2.1.5 Project Documents

See Section 8.2.1.5 of the *PMBOK® Guide*.

Additional considerations include the following:

- 0Inputs for performing software product and process software quality assurance include the release plan and test plan, project or organizational procedures, project records and test result records, reports of design and code reviews, inspections, and audits. Automated tools for requirements management, software configuration management, release management, and problem management are common sources of records for an SQA review and audit. In some software projects, there may be documented records demonstrating that the planned efforts occurred. In other cases, those who are responsible for quality assurance personally witness the steps to satisfy themselves that the process is working as planned. On small projects, it may be the development manager or the project manager who assumes this responsibility.

- 0In today's software projects, the SQA team participates during analysis to define acceptance criteria and test plan details ahead of development. The test plans themselves become part of the requirements communicated to the delivery team. Other inputs for SQA are various analytical simulations that predict the most likely number of defects to be expected in the code, based on previous test results, the complexity of the software, the experience of the developers, and other factors. When this analysis has been performed, the results are inputs to performing SQA. This information is used to check the validity of the tests results.

- 0For software projects that use adaptive life cycles, the details of test plans, including specific acceptance criteria are progressively elaborated along with the requirements. Feature-level criteria are developed as part of the analysis and design at the feature level. Detailed story-level acceptance criteria are defined as part of making the requirements backlog ready for the development team. This means that the SQA team is continuously involved with the development team from analysis to acceptance of the delivered increments of software.

- 0Inputs also include work performance data, such as work effort and elapsed time and cost to date, because they will be compared to the project's plans in order to measure the variance between the plans and the actual results; burn-down charts are analyzed for adaptive life cycle projects. By doing this type of comparison at frequent intervals, the project is able to determine where changes may be necessary to procedures or to schedules. Thus, the quality of planning is improved throughout the project.

8.2.2 Perform Software Project Quality Assurance: Tools and Techniques

The tools and techniques for performing quality assurance in Section 8.2.2 of the *PMBOK® Guide* are applicable to performing quality assurance for software projects. Additional considerations include the following.

For predictive software project life cycles, specialized SQA personnel who are independent of the development process typically conduct SQA activities. That is, developers do not perform acceptance testing on their own work and those responsible for SQA do not report directly to the project manager or software development lead. For predictive software projects, SQA budgets are usually not controlled by the development manager.

For safety-critical projects, SQA is sometimes conducted by an external group, that is chartered to ensure that an organization's software projects and products meet the organization's and customer's standards throughout the software project life cycle. These activities verify the extent to which quality objectives were met, and the effectiveness of the quality control methods and activities are assessed.

In adaptive life cycle projects, internal SQA activities are conducted by the project team itself through techniques such as pair programming, peer reviews5, functional testing, and demonstrations of working software within the development team. Iterative-incremental software projects typically use a blend of these approaches; story-level testing is part of the development team process while feature-level and release-level testing are conducted by an external testing group.

Quality auditors compare actual processes to documented processes by observation and inspection of records. Quality auditors may discover a lack of documentation or erroneous documentation that needs to be updated. As described in Section 8.2 of the *PMBOK® Guide*, QA personnel audit the results from quality control measurements to assess whether or not the quality requirements are being met. If not, the software project manager assures that action is taken to correct the discrepancies. While adaptive life cycle teams value working, deliverable software over documentation, some level of documentation is necessary to meet internal and external quality requirements. Quality auditors ensure that project teams are meeting the necessary level of working, deliverable software and the supporting documentation.

SQA also includes examination of the volatility of software product requirements. Frequent changes to requirements can be a warning that there are serious problems in the project. This may indicate that the system boundaries are not well defined, or that affordability constraints need to be addressed by adjusting the scope of product features for a release. Note, however, that emerging or derived requirements (i.e., those that further elaborate high-level requirements) are not classified as symptoms of volatility. These are simply refinements. An adaptive development project may begin work on one set of features that is sufficiently well understood even if other requirements are still unknown or changing, or the project team may produce a prototype to allow users to agree whether the approach will fulfill their quality expectations.

The practice of code reviews (i.e., walkthroughs and inspections), sometimes called static testing, is one of the most cost-effective methods to remove defects early in the development process and assure code quality. Another technique that supports software quality assurance is frequent testing. Software builds (often on a daily or even continuous basis) integrate changes to the software and are followed by smoke tests.

Usability evaluations, in the form of heuristic evaluations or cognitive walkthroughs, are cost effective at finding defects that might later on require reworking the software. Structured video-recorded usability testing with user representatives, using a "think-aloud" approach, is also useful for finding defects well before release to the end-user.

Quality assurance is integrated into adaptive software project life cycles by the inclusion of retrospective reviews on each iterative cycle. The retrospective review is used to assess the results of the completed iteration and to plan for process improvements in successive iterations. Retrospectives can sometimes result in finger pointing or may occur at times when the developers are feeling rushed due to insufficient time to produce features. An important way to overcome any frustration is to make sure that the software developers are involved in the specification of the release criteria, so that they

understand the goals and will ensure that planning for the next iteration incorporates the needed process changes. This approach contributes to team learning and builds continuous improvement into the project iterations.

Process analysis is a foundational element of process improvement. Widely used in other areas of business, process analysis (and process flow diagramming) helps software project teams to gain and increased understanding of how they can work together in a more efficient and more effective manner. Focusing on the throughput of software features or maintenance changes is well suited to process improvement efforts that identify bottlenecks, process delays, and sources of error. Tools such as flowcharts and process flow diagrams, as well as state charts, originally used for designing programs, can be used as process improvement tools to document process flows and process state transitions. For example, the various states that a defect passes through from first being reported until being resolved may be depicted in a flow diagram or a state-transition diagram.

Training is covered in Section 10 (Human Resources); however, training may also be regarded as a quality assurance technique, particularly the training required for individual software development processes and team software processes.

8.2.2.1 Quality Management and Control Tools

See Section 8.2.2.1 of the *PMBOK® Guide*.

8.2.2.2 Quality Audits

See Section 8.2.2.2 of the *PMBOK® Guide*.

8.2.2.3 Process Analysis

See Section 8.2.2.3 of the *PMBOK® Guide*.

8.2.3 Perform Software Project Quality Assurance: Outputs

The outputs for performing quality assurance in Section 8.2.3 of the *PMBOK® Guide* are applicable to performing quality assurance for software projects.

8.2.3.1 Change Requests

See Section 8.2.3.1 of the *PMBOK® Guide*.

8.2.3.2 Project Management Plan Updates

See Section 8.2.3.2 of the *PMBOK® Guide*.

8.2.3.3 Project Documents Updates

See Section 8.2.3.3 of the *PMBOK® Guide*.

8.2.3.4 Organizational Process Assets Updates

See Section 8.2.3.4 of the *PMBOK® Guide*.

Additional considerations include the following.

As described in Section 8.2.3 of the *PMBOK® Guide*, the outputs of the Quality Assurance process are audit reports and change requests that provide inputs to the Perform Integrated Change Control process (see Section 4.5 of the *PMBOK® Guide* and of this software extension). Audit reports and change requests may also show the need for changes in software project planning. These changes will be reflected in the software project team's process and product documented work activities.

- 0**Cost of Corrective Rework.** The cost of corrective rework for software is often a significant percentage of the total cost of developing a software product. However, these costs are often not known. For example, in projects using test-driven development, internal defects may be significantly reduced during development, but because defects are found and addressed the same day, thereby reducing later rework, the cost is not separately identified. Nonetheless, it is well known that investment in quality improvement by software organizations to prevent (or reduce) the occurrence of defects reduces rework and often results in impressive returns on investment, in addition to improvements in intangible factors, such as team morale and customer satisfaction. The intangible costs of poor quality include the loss of reputation that accrues to late delivery and poor quality of a delivered software product. Also, constant rework to fix defects can impact the morale of software developers.

- 0**Technical Debt.** The concept of technical debt is closely related to the cost of quality. For predictive, plan-driven, or time-driven software projects, defects are often not discovered (or are not corrected) near the point of injection. They become exponentially more expensive to fix the longer they persist. It is not uncommon in such projects that a requirements defect not found until systems testing may cost, in time and effort, 100 times more to fix than it would cost to find and fix it during a requirements review. Fixing customer-discovered defects becomes even more expensive. Failure to find and fix defects early and deferring the fixing of known defects creates technical debts that are repaid later by the cost incurred for corrective rework. Accumulation of technical debt is sometimes referred to as "mortgaging the future"; the interest rate on the mortgage can be excessive in linear, plan-driven projects. Projects strive to minimize technical debt by focusing on and correcting defects throughout the development process without incurring significant costs.

8.3 Control Software Project Quality

Section 8.3 of the *PMBOK® Guide* states that Control Quality is the process of monitoring and recording results for executing the quality activities to assess performance and recommend necessary changes. SQC is a system of routine technical activities used to measure and control the quality of the development processes and the quality of the product as it is being developed.

The most effective method of controlling and improving software quality is to focus on early detection and removal of software defects using continuous verification and validation techniques and to focus on changing the software development process to improve defect prevention. A large part of quality control for software products has historically relied on a predictive approach of post-development techniques, including staged levels of software testing and analysis of the detected defects. Adaptive software project life cycles integrate testing and demonstration of working, deliverable software on repetitive cycles throughout the software development process.

Software quality control (SQC) includes operational techniques and activities to verify that requirements for quality are being met and to take action when they are not. SQC is defined as a set of activities that evaluate and report the quality of software project artifacts throughout the project life cycle. For purposes of this standard, "evaluate the quality" means comparing work products to their requirements (including agreements, plans, policies, requirements, and designs). SQC often relies on statistical methods such as control charts and Pareto diagrams to analyze software defects and the associated rework

to correct the defects. Interventions may be required to prevent the release of defective software. This may involve feedback for process correction or process improvement.

8.3.1 Control Software Project Quality: Inputs

The inputs for controlling quality in Section 8.3.1 of the *PMBOK® Guide* are applicable to controlling quality for software projects. As described in the *PMBOK® Guide*, inputs to software quality control include management plans and checklists. Another significant input is the measurement plan for the prioritized quality attributes as defined in the release criteria. Project records, especially test records and CM records, are essential inputs and are typically maintained in controlled repositories.

8.3.1.1 Project Management Plan

See Section 8.3.1.1 of the *PMBOK® Guide*.

8.3.1.2 Quality Metrics

See Section 8.3.1.2 of the *PMBOK® Guide*.

8.3.1.3 Quality Checklists

See Section 8.3.1.3 of the *PMBOK® Guide*.

8.3.1.4 Work Performance Data

See Section 8.3.1.4 of the *PMBOK® Guide*.

8.3.1.5 Approved Change Requests

See Section 8.3.1.5 of the *PMBOK® Guide*.

8.3.1.6 Deliverables

See Section 8.3.1.6 of the *PMBOK® Guide*.

8.3.1.7 Project Documents

See Section 8.3.1.7 of the *PMBOK® Guide*.

8.3.1.8 Organizational Process Assets

See Section 8.3.1.8 of the *PMBOK® Guide*.

8.3.2 Control Software Project Quality: Tools and Techniques

The tools and techniques for controlling quality in Section 8.3.2 of the *PMBOK® Guide* are applicable to controlling quality for software projects (see Sections 8.3.2.1 – 8.3.2.4). Additional considerations include the following.

The basic quality tools of statistical process control, including control charts, run

charts, histograms, and Pareto diagrams, can be applied to the analysis of defect patterns in software, thus providing the basis for identify areas for preventative actions.

Along with those items listed in the *PMBOK® Guide*, techniques for SQC include reviews, testing, and the version control elements of configuration management. Reviews take many forms, such as code walkthroughs, design inspections, and reviews of other work products, for example, user manuals and installation instructions. Static analysis and testing tools are typically used. Of the QC tools and techniques identified in the *PMBOK® Guide*, inspection (also called examination or walkthrough) is one of the most effective ways to identify software defects and omissions in software and documentation. The reviews may be manual or they may use specific tools that check for common coding errors or spelling errors. Defects are corrected when found, thereby controlling the quality of the work products.

Test-driven development has long proved its use in verifying the quality of software. In this approach, test cases are written *before* writing any functional code, and the test cases are run to demonstrate that they fail. Then the new code is added, after which the test cases are run again to demonstrate that they no longer fail. Tools are commonly used to automate such testing, but these can be desk exercises, by walking through the code.

Software testing includes unit testing, integration and verification testing, validation and acceptance testing, and regression testing. Often, development teams will build scaffolding, or temporary modules, to support early testing, simulating inputs from other (nonexistent) parts of the system or the integration of various modules that have not yet been constructed. This allows integration or regression testing to be performed at an early stage in software development. Testing may also focus on specific quality attributes, such as performance testing, load testing, security testing, or usability testing. User observation, formalized as usability testing, and user surveys measure quality-in-use characteristics, such as user satisfaction or efficiency.

Testing tools and scripts can execute repeat tests consistently with little or no manual intervention, automatically collect and store the test results, and refresh the data for another round of testing. Testing tools may save time for the test team to focus on the design of tests and analysis of the results.

For some types of systems or domains, software quality can be further improved by automatically generating it from specifications or higher-level languages, where possible. This reduces the amount of error-prone coding. Model-driven development involves generating substantial parts of software from “models” written in languages such as UML and/or domain-specific languages.

Configuration management (CM) also plays a significant role in improving quality during software development. CM provides routine and consistent checks to ensure the integrity, correctness, and completeness of each software build, often based on execution of scripts. Enforced configuration control avoids the problems that arise if more than one developer works on the same branch of code at the same time. Configuration management is usually automated by several tools working together. Control of versions of each source file document, script and the “configurations” or sets of these that belong together is accomplished using configuration management tools, available as free open-source or as commercial products. These tools can also manage the different ways that components are put together to create different end-products in a software product line or software product family. Tools are also used to track the defects and other issues related to the software, and the resolution of these issues. ISO/IEC/IEEE Standard 828 – Configuration Management in Systems and Software Engineering **[32]** provides guidance on all aspects of configuration control for systems and software.

SQC can also identify record, analyze, and treat software defects. Software defects may be classified for severity (i.e., the impact on the user), urgency (i.e., the importance to users, often designated as “priority”), root cause of the defect, or location of the defect the code. In addition, defect find/fix data provides statistical data to assess the

level of stability or instability of a software system at a point in time.

SQC uses most of the quality tools described in the *PMBOK® Guide*, particularly control charts, run charts, Pareto charts, and histograms. These charts help the software manager to visualize extensive data and discern patterns and causes.

Control charts are one of the most common tools used in software projects, even though they are not usually called control charts. These charts are used to track information such as numbers of open defects by severity, numbers of changes to the code, and numbers of changes to requirements. In these cases, the lower limit is zero, and the upper limits are specified in the release criteria.

Run charts are often used for defect analysis. The numbers of defects found each week (or day) are plotted with their dates. A chart can present all defects along with a breakdown of defects by severity. The chart will show trends, such as declining numbers of defects found as the product gains stability. By recording the number of defects identified and corrected each day, the SQC team can see the rate at which defects are being fixed. If there is a significant number of unresolved defects, the project manager and stakeholders need to decide if the project duration will be extended until the issues are resolved, or if the defects will carry over to the next release, which will reduce the intake of new work from the requirements backlog.

Pareto charts can be used to show the distribution of defects across product components. Those components that show high levels of defects (error-prone components) may require a design or code review by senior members of the team to determine the cause of the problems. Pareto charts are also used to graph data from software configuration management. Software components that are changing excessively may indicate a dangerous kind of code volatility, for example, a situation in which each defect "fix" breaks some other part of the code.

Histograms are useful in identifying process failures. For example, when software builds are failing frequently over time, it is necessary to investigate. By keeping track of reasons why the builds fail, the changes that need to be made can be identified. In the case of software build failure, it is often determined that the build process hasn't been automated and that the checklist needed for successful manual operation is incomplete, wrong, or missing. Similarly, if regression tests fail, the cause may be that a "fix" broke something else, earlier fixes were faulty, or possibly that the wrong code was compiled.

8.3.2.1 Seven Basic Quality Tools

See Section 8.3.2.1 of the *PMBOK® Guide*.

8.3.2.2 Statistical Sampling

See Section 8.3.2.2 of the *PMBOK® Guide*.

8.3.2.3 Inspection

See Section 8.3.2.3 of the *PMBOK® Guide*.

8.3.2.4 Approved Change Requests Review

See Section 8.3.2.4 of the *PMBOK® Guide*.

8.3.3 Control Software Project Quality: Outputs

The outputs for controlling quality in Section 8.3.3 of the *PMBOK® Guide* are applicable to controlling quality for software projects (see Section 8.3.3.1 – 8.3.3.8). Additional considerations include the following.

The outputs of quality control presented in the *PMBOK® Guide* are applicable for software projects:

- 0Measurements of the quality attributes specified in the quality management plan and the release criteria.
- 0Changes to software and other artifacts validated by testing or inspection.
- 0Deliverables validated by testing or inspection to conform to the scope identified at the start of the project or iteration.
- 0Identification of gaps between planned and actual performance and reasons for the gaps.
- 0Updated checklists, tests, and other process assets.
- 0Lessons learned by means of the project or iteration retrospective, along with the team's recommendations for changes in the process or product, and the resultant change requests.
- 0Updates to the management plans (e.g., schedule, resources, configuration management, test planning).

8.3.3.1 Quality Control Measurements

See Section 8.3.3.1 of the *PMBOK® Guide*.

8.3.3.2 Validated Changes

See Section 8.3.3.2 of the *PMBOK® Guide*.

8.3.3.3 Verified Deliverables

See Section 8.3.3.3 of the *PMBOK® Guide*.

8.3.3.4 Work Performance Information

See Section 8.3.3.4 of the *PMBOK® Guide*.

8.3.3.5 Change Requests

See Section 8.3.3.5 of the *PMBOK® Guide*.

8.3.3.6 Project Management Plan Updates

See Section 8.3.3.6 of the *PMBOK® Guide*.

8.3.3.7 Project Documents Updates

See Section 8.3.3.7 of the *PMBOK® Guide*.

8.3.3.8 Organizational Process Assets Updates

See Section 8.3.3.8 of the *PMBOK® Guide*.

9 SOFTWARE PROJECT HUMAN RESOURCE MANAGEMENT

As stated in the *PMBOK® Guide*, Project Human Resource Management includes the processes that organize, manage, and lead the project team. The *PMBOK® Guide* provides excellent general purpose Human Resource Management advice that is suitable for a variety of project environments. It covers how to acquire appropriate project resources, develop them, and manage them from a domain independent viewpoint. It can be applied to machinists, construction workers, or researchers.

However, because of the need to provide universal guidance applicable on any project, the *PMBOK® Guide* does not focus on domain specific help for knowledge worker projects. Software projects are staffed by knowledge workers who collaborate to solve problems in novel environments with incomplete data. To most effectively manage these resources it is appropriate to utilize the knowledge worker recommendations of Peter Drucker, and Don Reinertson. **[33]**.

Software project team members typically possess technical knowledge and skills superior to the project managers managing them. Therefore, to be most effective, project managers need to find ways to tap into and leverage the knowledge and skills of software project team members. Successful software project managers typically put less emphasis on directing the work and more on facilitating the efficiency and effectiveness of project teams. This subtle, but crucial shift dramatically changes the way teams are created, developed, and managed. It effectively changes the approach to “Plan Human Resource Management” and “Manage Project Team(s).”

Also, since software teams spend a large proportion of their time collaborating, discussing ideas and making joint decisions, the “fit” of participants within the team is extremely important. Rather than hiring a competent programmer who can do good work in isolation, the ease and effectiveness of interacting with the members of the software team can be as important as their technical ability. So it is desirable to have team members engaged in the selection process of other team members. This influences the “Acquire Project Team” roles on a project.

Software project teams often build novel solutions using new technology, therefore, they will not have the necessary solutions available upfront on the project. Instead they are encouraged to innovatively solve problems, iterate on proofs of concepts, and improve their processes as they develop the software product. This leads to the desirability of self-empowered teams who self diagnose, engage in retrospection, and continuously improve. The process of instilling and promoting these ideas is common in successful software projects and influences the way we develop and manage project teams.

These changes to how we Plan Human Resource Management, Acquire Project Teams, Develop Project Team, and Manage Project Teams are described in the following sections of this software extension.

Figure 9-1 provides an overview of Software Project Human Resource Management; it is an adaptation of Figure 9-1 in the *PMBOK® Guide*.

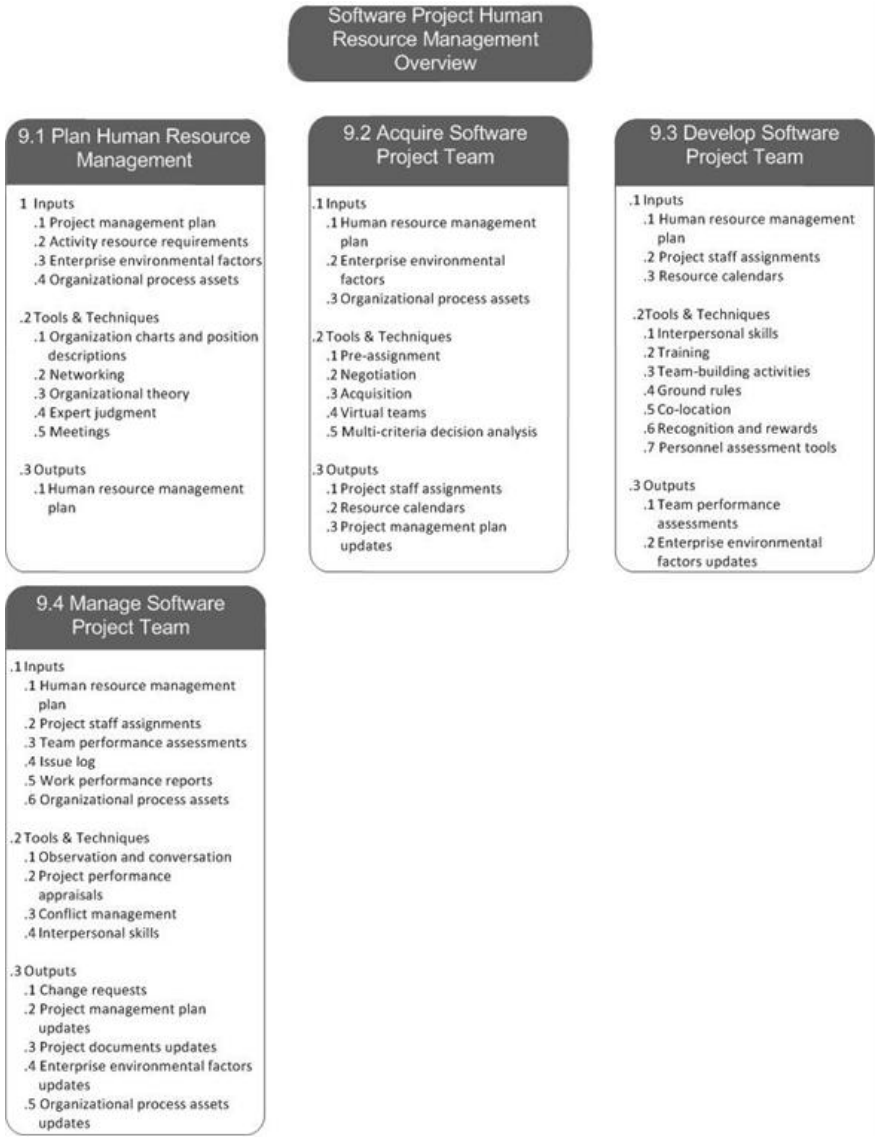


Figure 9-1. Software Project Human Resource Management Overview

9.1 Plan Human Resource Management

The Plan Human Resource Management process involves identifying and documenting project roles, responsibilities, required skills, reporting relationships, and creating a staffing management plan. The inputs, tools and techniques, and outputs in Section 9.1 of the *PMBOK® Guide* are applicable to planning human resource management for software projects.

9.1.1 Plan Human Resource Management: Inputs

The inputs for planning human resource management in the *PMBOK® Guide* are applicable to planning human resource management for software projects.

9.1.1.1 Project Management Plan

See Section 9.1.1.1 of the *PMBOK® Guide*.

9.1.1.2 Activity Resource Requirements

See Section 9.1.1.2 of the *PMBOK® Guide*.

9.1.1.3 Enterprise Environmental Factors

See Section 9.1.1.3 of the *PMBOK® Guide*.

9.1.1.4 Organizational Process Assets

See Section 9.1.1.4 of the *PMBOK® Guide*.

9.1.2 Plan Human Resource Management: Tools and Techniques

The tools and techniques for planning human resource management in the *PMBOK® Guide* are applicable tools and techniques for planning human resource management for software projects.

9.1.2.1 Organization Charts and Position Descriptions

See Section 9.1.2.1 of the *PMBOK® Guide*.

9.1.2.2 Networking

See Section 9.1.2.2 of the *PMBOK® Guide*.

9.1.2.3 Organizational Theory

See Section 9.1.2.3 of the *PMBOK® Guide*.

9.1.2.4 Expert Judgment

See Section 9.1.2.4 of the *PMBOK® Guide*.

9.1.2.5 Meetings

See Section 9.1.2.5 of the *PMBOK® Guide*.

9.1.3 Plan Human Resource Management: Outputs

The output for planning human resource management in the *PMBOK® Guide* (human resource plan) is applicable for planning human resource management for software projects.

9.1.3.1 Human Resource Management Plan

See Section 9.1.3.1 of the *PMBOK® Guide.*

For software projects, planning human resource management occurs with recognition of some additional software project truisms and characteristics.

Software projects require collaboration and information sharing to solve problems and build new products. Team members are motivated by opportunities to expand their skills, solve interesting problems, build innovative software, and use effective software tools. Failing to recognize the motivational factors of software developers when planning HR management can create many problems later on in a project.

Software teams perform better with less of a command-and-control structure and more of a facilitation approach to project management **[34]**. Instead of planning to give detailed task lists to team members, effective software project managers plan on presenting work as questions or problems to be solved and letting team members organize as they see best to meet these challenges. Not only does this provide a more stimulating and rewarding environment for software teams, it also defers design decisions and keeps solutions open to creative solutions that may not have been envisioned during the initiation and planning phases of a software project.

A commonly used approach to managing software projects is to adopt a servant leadership role by the project manager, which enables empowered teams. Team members are encouraged to wear many hats and pitch-in regardless of their formal title, which balances the workload and enables completion of the tasks required for project success.

In recognition of the professionalism generally seen on software projects, project managers are encouraged to take more of a Theory Y view of team members and less of a Theory X approach **[35]**.

McGregor's Theory X asserts that employees are inherently lazy and will avoid work whenever they can. Theory X managers believe that workers need to be closely supervised and comprehensive systems of controls must be developed and enforced. Theory Y posits that employees are ambitious and self-motivated. They enjoy creative problem solving, but their talents are underused in most organizations. Theory Y managers communicate openly with team members, minimize the difference between superior-subordinate relationships, and create a comfortable environment in which subordinates can develop and use their talents and abilities. This climate includes shared decision making so that subordinates have a say in decisions that influence them and their work products **[36]**.

So, when planning Software Project Human Management, software project managers modify the approach for the characteristics of software project workers, try to avoid command and control structures, and instead tap into the problem-solving motivational factors that drive software professionals. Also, software project managers plan for cross-functional teams with all the skills needed to deliver the product represented on the team.

9.2 Acquire Software Project Team

The inputs, tools and techniques, and outputs for acquiring a project team in Section 9.2 of the *PMBOK® Guide* are applicable to acquiring a software project team.

9.2.1 Acquire Software Project Team: Inputs

The inputs for acquiring a project team in the *PMBOK® Guide* are applicable to acquiring a project team for software projects.

9.2.1.1 Human Resource Management Plan

See Section 9.2.1.1 of the *PMBOK® Guide.*

9.2.1.2 Enterprise Environmental Factors

See Section 9.2.1.2 of *the PMBOK® Guide.*

9.2.1.3 Organizational Process Assets

See Section 9.2.1.3 of the *PMBOK® Guide*. In addition, it is to be observed that software organizations sometimes acquire software project team members by hiring contract personnel to perform various project duties. Contract personnel may be members of the software development team or they may be hired to perform specialized tasks such as design or testing. These contracted personnel do not always have allegiance to the organization or the project and may not adapt readily to the corporate culture of the organization.

9.2.2 Acquire Software Project Team: Tools and Techniques

The tools and techniques for acquiring a project team in the *PMBOK® Guide* are applicable to acquiring a project team for software projects.

9.2.2.1 Pre-assignment

See Section 9.2.2.1 of the *PMBOK® Guide*.

9.2.2.2 Negotiation

See Section 9.2.2.2 of the *PMBOK® Guide*.

9.2.2.3 Acquisition

See Section 9.2.2.3 of the *PMBOK® Guide*.

9.2.2.4 Virtual Teams

See Section 9.2.2.4 of the *PMBOK® Guide*.

9.2.2.5 Multi-Criteria Decision Analysis

See Section 9.2.2.5 of the *PMBOK® Guide*.

9.2.3 Acquire Project Team: Outputs

The outputs for acquiring a project team in the *PMBOK® Guide* are applicable for acquiring project teams for software projects.

In addition, the following considerations address some particular aspects of acquiring a software project team.

Since software project team members share and manipulate information rather than tangible materials, team stability and dedicated team members are important attributes that reduce reiterating the goals, the agreed-to approach, and the mechanisms for determining project status that occurs if there is turnover in the team. Software project teams work best within strong matrix and projectized environments, where a dedicated team can work on a single project with few interruptions. Team members work together most effectively when they are co-located and face-to-face communications can occur on a continuous, ongoing

basis.

The goal of acquiring software project team members is to create stable, co-located teams that have all of the skills needed to conduct the project. Silo teams with matrix reporting structures are less likely to be committed to shared project goals since some of their allegiance will be to their host group. If colocation is not possible, stable teams in different time zones are preferred to silo teams.

Involvement of present team members, in addition to the project manager, when acquiring new team members also increases the likelihood of building an integrated team. HR and project management representatives do the normal front-end screening of candidates to weed out unsuited or unskilled applicants. The acceptable candidates are then invited for peer interviews where team members assess the candidate to determine whether they can work with the candidate, whether the candidate will be a good fit for the team, and whether the candidate will make the team stronger or weaker. Of course, care must be taken to ensure that new team members will bring diversity of viewpoints and fresh thinking to the team.

By engaging different software groups, for example, software developers and QA staff, different characteristics of the candidate will be evaluated. Everyone wins when engaging the team in the interview process. The project team wins because they have already met and endorsed the prospective candidate. Candidates win because they get to meet their potential peers, learn what the project is about, and are better able to assess the corporate culture. The project manager wins because the team members, who have the technical knowledge, will ask the appropriate technical questions. Finally, all subgroups within the project win by learning what characteristics are important to other subgroups within the project, thereby increasing their awareness and maturity.

9.2.3.1 Project Staff Assignments

See Section 9.2.3.1 of the *PMBOK® Guide*.

9.2.3.2 Resource Calendars

See Section 9.2.3.2 of the *PMBOK® Guide*.

9.2.3.3 Project Management Plan Updates

See Section 9.2.3.3 of the *PMBOK® Guide*.

9.3 Develop Software Project Team

The inputs, tools and techniques, and outputs for developing the project team in Section 9.3 of the *PMBOK® Guide* are applicable to developing a software project team.

9.3.1 Develop Software Project Team: Inputs

The inputs for developing a project team in the *PMBOK® Guide* are applicable to developing a project team for software projects.

9.3.1.1 Human Resource Management Plan

See Section 9.3.1.1 of the *PMBOK® Guide*.

9.3.1.2 Project Staff Assignments

See Section 9.3.1.2 of the *PMBOK® Guide.*

9.3.1.3 Resource Calendars

See Section 9.3.1.3 of the *PMBOK® Guide*.

9.3.2 Develop Software Project Team: Tools and Techniques

The tools and techniques for acquiring a project team in the *PMBOK® Guide* are applicable to acquiring a project team for software projects.

9.3.2.1 Interpersonal Skills

See Section 9.3.2.1 of the *PMBOK® Guide*.

9.3.2.2 Training

See Section 9.3.2.2 of the *PMBOK® Guide*.

9.3.2.3 Team-Building Activities

See Section 9.3.2.3 of the *PMBOK® Guide*.

9.3.2.4 Ground Rules

See Section 9.3.2.4 of the *PMBOK® Guide*.

9.3.2.5 Colocation

See Section 9.3.2.5 of the *PMBOK® Guide*.

9.3.2.6 Recognition and Rewards

See Section 9.3.2.6 of the *PMBOK® Guide*.

9.3.2.7 Personnel Assessment Tools

See Section 9.3.2.7 of the *PMBOK® Guide*.

9.3.3 Develop Software Project Team: Outputs

The outputs from Develop Project Team in the *PMBOK® Guide* are applicable to developing a software project team (see Sections 9.3.3.1 and 9.3.3.2).

The following considerations address some additional aspects of developing a software project team.

Develop Software Project Team is concerned with the process of improving the competencies, team interactions, and the overall team environment to enhance project performance. On software projects this is a nested, recurring pattern that happens continuously on cycles

of exploration and feedback that typically occur on hourly, daily, weekly, biweekly, and monthly iteration cycles.

The practice of pair programming, where two software developers share a task, can assist greatly with skills improvement and learning of good practices. Often, team members of dissimilar skill levels are paired (senior with junior) and pair members are rotated frequently in order to maximize learning opportunities. This also has the benefit of sharing project and technical knowledge throughout the team, and reducing the dependency on key individuals for their knowledge. If a team member happens to leave the project, the impact is not as significant because they are others who understand the topic.

The practice of test driven development (TDD) also helps improve team competencies through short feedback cycles of experimental learning. TDD or "red, green, refactor" refers to the steps of writing a test (that fails), then writing code until the test passes, then refactoring the code for clarity; this process may occur many times each day. By encouraging developers to think about how code will be tested before writing code, business purpose and usability are considered frequently, which enhances quality and acceptance. However, the main benefit is for the team, who will learn through rapid cycles of exploration, test, and feedback. These concepts, as they are reinforced in adaptive software project life cycle projects, are illustrated in Figure 9-2.

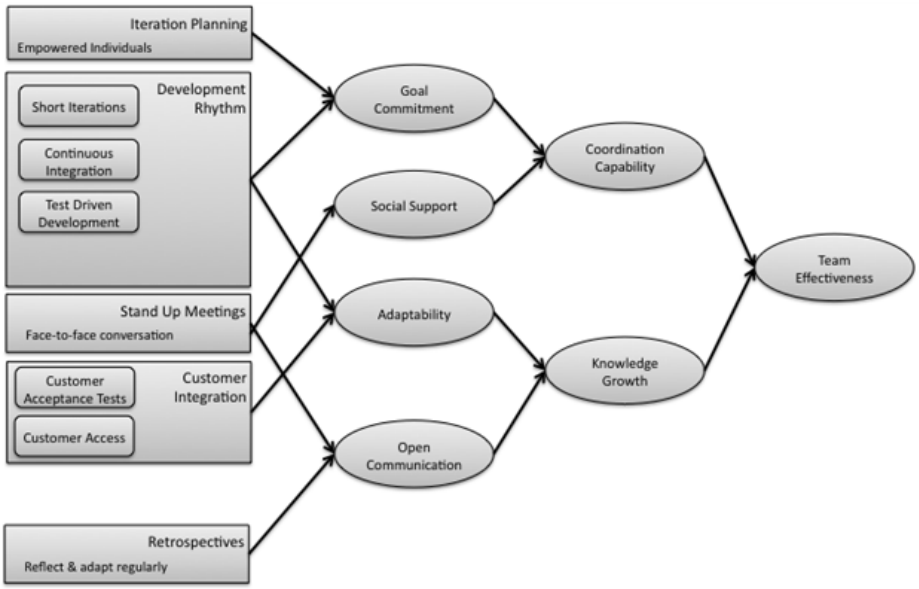


Figure 9-2. Attributes that Increase Software Project Team Effectiveness

Software project teams coalesce and become more productive if they are stable and co-located. It takes time for teams to progress through the Tuckman stages of forming, storming, norming, and finally to performing to optimize team output **[38]**. Swapping people in and out of a team triggers the storming and norming phases again as new team members find their place in the team and the team adjusts to them.

Part of the storming and norming process for software project team members is learning how to deal with team conflict, gaining commitment for decisions, and ultimately developing a sense of accountability for the project outcomes. These are complex issues that impact all projects where skilled people need to collaborate on building novel solutions; these are particularly important issues for software project teams. Getting skilled people to work together and harness constructive disagreement and rigorously test decisions is a primary goal of software team development.

Colocation of team members helps this process and allows direct face-to-face communication. Colocation is not always possible, but given a choice of two teams, one experienced but dispersed, and one more junior but local, the local team is often the best choice for a software project.

The challenges software teams face in working together and learning to trust one another, thrash out issues, make decisions, and commit to shared responsibility are well described by *The Five Dysfunctions of a Team* **[39]**:

- 0**Absence of trust.** Unwillingness to be vulnerable within the group. People must be open about mistakes and weaknesses to build a foundation of trust.
- 0**Fear of conflict**. Teams that lack trust cannot engage in unfiltered debate. Instead they resort to veiled discussions and guarded comments.
- 0**Lack of commitment**. Without passionate debate, team members rarely if ever buy-in to and commit to decisions, though they may feign agreement during meetings.
- 0**Avoidance of accountability**. Due to lack of commitment and buy-in, most people will hesitate to constructively confront their peers on actions and behaviors that seem counterproductive to the good of the team and the project.
- 0**Inattention to results**. Failure to hold one another accountable leads to putting individual goals (or department goals) ahead of the project.

These dysfunctions are ever-present risks for software teams. Project managers can help build a culture of trust and acceptance by sharing their mistakes with the team and demonstrating that it is OK to admit to mistakes.

Colocation also helps to facilitate unfiltered debate. Without the barriers of video conferencing, email, and telephone, it is much easier to get to the heart of an issue when in direct communication. Empowered teams and shared decision-making helps to build commitment.

Daily stand-up meetings, iteration planning meetings, and retrospectives at the ends of iterations reinforce and reiterate team member work commitments and team accountability for results.

Self-organizing teams schedule their own work and take ownership for problems and solutions wherever possible. For some teams, this may come naturally; for others, it will be a big transition that requires training and encouragement by senior management.

9.3.3.1 Team Performance Assessments

See Section 9.3.3.1 of the *PMBOK® Guide*.

9.3.3.2 Enterprise Environmental Factors Updates

See Section 9.3.3.2 of the *PMBOK® Guide*.

9.4 Manage Software Project Team

The process of managing a project team in Section 9.4 of the *PMBOK® Guide,* which covers tracking team member performance, providing feedback, resolving issues, and managing

changes to optimize project performance, is applicable to managing a software project team. The inputs, tools and techniques, and outputs in Section 9.4 of the *PMBOK® Guide* are applicable to managing a software project team.

9.4.1 Manage Software Project Team: Inputs

The inputs for managing a project team in the *PMBOK® Guide* are applicable to managing a software project team.

9.4.1.1 Human Resource Management Plan

See Section 9.4.1.1 of the *PMBOK® Guide*.

9.4.1.2 Project Staff Assignments

See Section 9.4.1.2 of the *PMBOK® Guide*.

9.4.1.3 Team Performance Assessments

See Section 9.4.1.3 of the *PMBOK® Guide*.

9.4.1.4 Issue Log

See Section 9.4.1.4 of the *PMBOK® Guide*.

9.4.1.5 Work Performance Reports

See Section 9.4.1.5 of the *PMBOK® Guide*.

9.4.1.6 Organizational Process Assets

See Section 9.4.1.6 of the *PMBOK® Guide*.

9.4.2 Manage Software Project Team: Tools and Techniques

The tools and techniques for managing a project team in the *PMBOK® Guide* are applicable to managing a software project team. They are observation and conversation, project performance appraisals, conflict management, and interpersonal skills.

9.4.2.1 Observation and Conversation

See Section 9.4.2.1 of the *PMBOK® Guide*.

9.4.2.2 Project Performance Appraisals

See Section 9.4.2.2 of the *PMBOK® Guide*.

9.4.2.3 Conflict Management

See Section 9.4.2.3 of the *PMBOK® Guide*.

9.4.2.4 Interpersonal Skills

See Section 9.4.2.4 of the *PMBOK® Guide*.

9.4.3 Manage Software Project Team: Outputs

The outputs from managing a project team in the *PMBOK® Guide* are applicable to managing a software project team.

9.4.3.1 Change Requests

See Section 9.4.3.1 of the *PMBOK® Guide*.

9.4.3.2 Project Management Plan Updates

See Section 9.4.3.2 of the *PMBOK® Guide*.

9.4.3.3 Project Documents Updates

See Section 9.4.3.3 of the *PMBOK® Guide*.

9.4.3.4 Enterprise Environmental Factors Updates

See Section 9.4.3.4 of the *PMBOK® Guide*.

9.4.3.5 Organizational Process Assets Updates

See Section 9.4.3.5 of the *PMBOK® Guide*.

The following considerations address some additional aspects of developing a software project team.

Tracking team member performance on a software project is a delicate issue. It is important to assess individual performance, interactions with colleagues, and development of skills. At the same time, care must be taken to not publicize measured performance at an individual level because many factors affect individual performance on a software project. For example, a talented project member may exhibit decreased productivity when working on the most complex part of the product. In addition, publicizing individual performance can result in self-centered behavior and provides little reward for collaborating with and helping other team members.

For these reasons, it is desirable to track performance at the team level; team members are incentivized to help colleagues in order to boost the team's overall productivity. For this reason "velocity" (the amount of work done per iteration) is measured at the team level, and not at the level of individuals.

Project managers who engage one-on-one with individual team members can learn each member's career development objectives. By developing the individual skills and roles of team members and then finding ways to build opportunities to use these skills on the project greatly improves individual commitment and satisfaction. Team members become more aligned and committed to the project goals when they see how their personal goals are linked to project execution.

Because many software projects work with short iterations, new roles can be tried for an iterative cycle or two before adopting or abandoning the new role. This opportunity to try new roles is appreciated by team members as being proactive to their needs without being

too disruptive to the project.

Regular intervals for experimentation are also advantageous to the project manager who rapidly obtains feedback on self-directed team adjustments. Iterative approaches provide short time periods for experimentation and feedback to team members, which most people find responsive and rewarding.

Feedback is obtained by demonstrating an increment of working software after which a retrospective team meeting is held. These two events provide valuable feedback to the project team members, project manager, and customer. The demonstration provides feedback on what the customer thinks of the new work, information is gained on how the project is meeting its goals, and retrospectives and introspection aids in adjusting and improving the development processes.

Resolving team issues and conflict also needs careful balance. Most team conflict is harmless and indeed positive, being a sign of a trusting environment where it is acceptable to present a dissenting view. Passionate debate over technical issues builds commitment to outcomes; conflict is only an issue when it extends beyond business and technical issues to become personal.

Because of the flexible nature of software, there is rarely only one way to solve a problem and so debate and discussion over approaches is normal and healthy as long as the discussions do not escalate beyond what the team can solve or overflow into personal conflict. If they do, the approaches to resolving conflict recommended in the *PMBOK® Guide* can be used.

10 SOFTWARE PROJECT COMMUNICATIONS MANAGEMENT

According to Section 10 of the PMBOK® Guide, Project Communications Management includes the processes that are required to ensure timely and appropriate planning, collection, creation, distribution, storage, retrieval, management, control, monitoring, and the ultimate disposition of project information. This section of the *Software Extension to the PMBOK® Guide – Fifth Edition* addresses project communication for software projects by addressing issues that are important for managing software project communication and that merit attention beyond that provided in the *PMBOK® Guide*.

The role of project communication is a primary consideration for software projects, because teams of individuals who engage in closely coordinated, intellectual activities develop software. With no physical product to reference, effective communication is paramount for keeping team members productively engaged and stakeholders informed. Software teams reduce complexity and enhance communication through a combination of communication approaches that include information radiators, colocation (when possible), and an emphasis on face-to-face communication.

Figure 10-1 provides an overview of Software Project Communication Management; it is an adaptation of Figure 10-1 in the *PMBOK® Guide*.

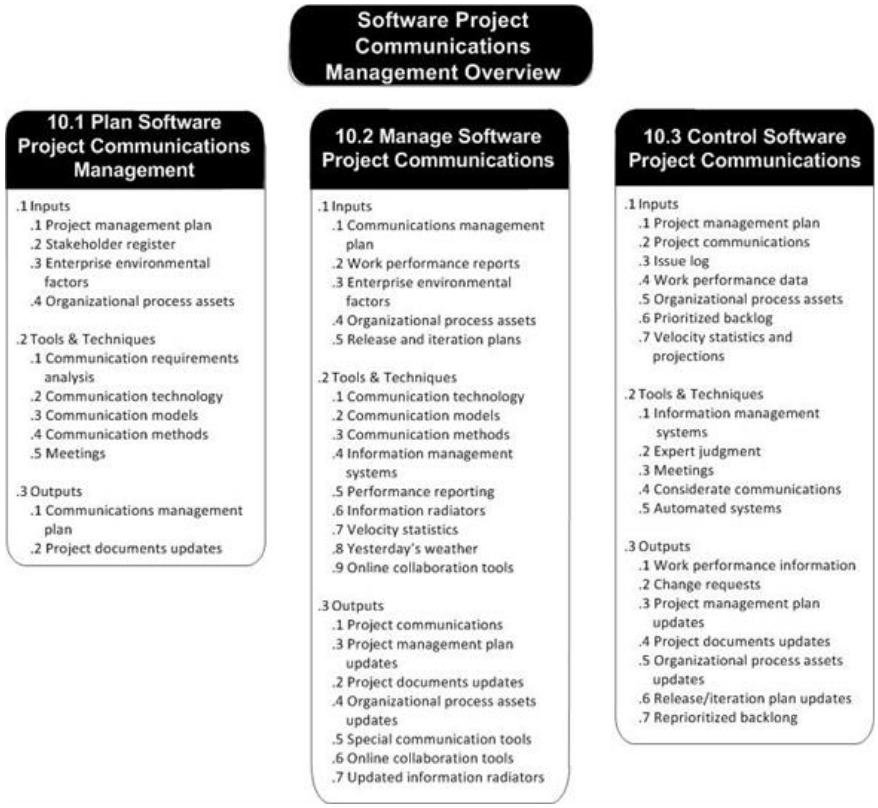


Figure 10-1. Software Project Communications Management Overview

10.1 Plan Software Project Communications Management

The inputs, tools and techniques, and outputs in Section 10.1 of the *PMBOK® Guide* are applicable to planning project communication management for software projects.

Software projects often exhibit high rates of change to accommodate changing and evolving requirements and shifting priorities. Frequent and productive communication among team members is important and can be achieved through daily stand-up meetings, frequent demonstrations of progress, and retrospective meetings. These approaches are typically, but not exclusively, applied to software projects that use adaptive project life cycles. Adaptive life cycles and the related communication techniques used may create confusion among stakeholders who are not familiar with them. In this case, the organization should plan extra time to explain the project life cycle processes. Additional communications and feedback cycles should be planned to ensure that all stakeholders understand why the project is operating this way, how team and other stakeholder communications works, and their involvement in the communication processes.

Face-to-face (FTF) communication allows two-way dialogs; issues and questions can be addressed immediately, and emotion is readily conveyed. For example, when discussing a topic, if someone begins nodding their head, indicating an understanding or agreement with the speaker, further explanation can be curtailed and the discussion can be moved on to other topics. Because of the higher bandwidth, opportunity for Q&A, and lower costs, face-to-face communication is the preferred means of communication for software development projects, whenever possible.

It is easy to productively use face-to-face communication for small, co-located teams. Video conferencing and voice-over-internet protocol (VOIP) technologies can be used to simulate face-to-face interactions if team members are geographically distributed.

To facilitate communication, the preferred solution for a growing team size is to break up large teams into multiple smaller sub-teams that can leverage face-to-face communication

and tacit knowledge within each of the smaller teams, with well-defined communication channels among the teams.

The following equation can be used to calculate the number of communication paths, P, among a collection of project teams, where n is the number of members per team and N is the number of teams. An assumption is made that each member of each project team communicates with all other members of their team, and one member of each project team communicates with all of the other project teams.

$$P = [n\ (n\text{-}1) / 2] + [N\ (N\text{-}1) / 2]$$

For a single team ($N = 1$) the number of communications paths is $P = n\ (n\text{-}1) / 2$; that is, communication paths within a project team increase on the order of the square of the number of team members.

Note, also, that a single team of 10 members has 45 communication paths, whereas two teams of 5 have 11 communication paths. Of course, the single point of contact for each team with the other team (i.e., the team leader) should have sufficient bandwidth to ensure effective communications between the two teams and among other teams when there are more than two teams.

10.1.1 Plan Software Project Communications: Inputs

The inputs in Section 10.1.1 of the *PMBOK® Guide* are applicable for planning software project communications. The following additional observation is also applicable.

Adaptive life cycles for software projects often include iteration and release plans for the iterations of the life cycle. These plans communicate the agreed-upon content for the next iteration and the content of the next iterative release (where a release may be used for a customer demonstration or for internal review by the project team). These plans provide an important input for planning software project communications.

10.1.1.1 Project Management Plan

See Section 10.1.1.1 of the *PMBOK® Guide*.

10.1.1.2 Stakeholder Register

See Section 10.1.1.2 of the *PMBOK® Guide*.

10.1.1.3 Enterprise Environmental Factors

See Section 10.1.1.3 of the *PMBOK® Guide*.

10.1.1.4 Organizational Process Assets

See Section 10.1.1.4 of the *PMBOK® Guide*.

10.1.2 Plan Software Project Communications: Tools and Techniques

The tools and techniques in Section 10.1.2 of the *PMBOK® Guide* are applicable for planning software project communications.

10.1.2.1 Communication Requirements Analysis

See Section 10.1.2.1 of the *PMBOK® Guide*.

10.1.2.2 Communication Technology

See Section 10.1.2.2 of the *PMBOK® Guide*.

10.1.2.3 Communication Models

See Section 10.1.2.3 of the *PMBOK® Guide*.

10.1.2.4 Communication Methods

See Section 10.1.2.4 of the *PMBOK® Guide*.

10.1.2.5 Meetings

See Section 10.1.2.5 of the *PMBOK® Guide*.

10.1.3 Plan Software Project Communications: Outputs

The in Section 10.1.3 of the *PMBOK® Guide* are applicable for planning software project communications.

10.1.3.1 Communications Management Plan

See Section 10.1.3.2 of the *PMBOK® Guide*.

10.1.3.2 Project Documents Updates

See Section 10.1.3.2 of the *PMBOK® Guide*.

Additionally, when planning communications management for a software project, it is important that the characteristics of software and knowledge work are recognized and incorporated. **[37]** These include:

- 0Software projects are often new or novel to their customers and host organizations; therefore, communication may be needed to explain the tools and techniques that will be used when managing the project, especially to manage ambiguity during the initiating and planning processes of a software development project.
- 0Software project life cycles are often complex, so significant communication is needed to explain the development process that will be used and the roles that various stakeholders will play.
- 0Software projects often experience high rates of change as product requirements evolve, so it is important that frequent communications are provided to keep stakeholders up to date. Communication mechanisms may include daily standup meetings, demonstrations of the evolving software product, and retrospectives to provide communication and coordination.
- 0Software projects are often undertaken by geographically dispersed teams, so electronic communication tools such as voice over internet protocol (VOIP), instant messaging, video conferencing, and project websites are often utilized.
- 0Both push (publish) and pull (subscribe) communication mechanisms are used to

accommodate the high rate of information exchange often seen in software projects.

Adaptive life cycles for software projects address these characteristics of project communication by frequently demonstrating the evolving functionality and regularly delivering functionality into the users' environment, if desired, to provide higher project visibility for most stakeholders. A major attribute of adaptive life cycles is to eliminate the long periods of internal project activity, which make it difficult for external stakeholders to understand what is happening.

Adaptive life cycle techniques facilitate the planning of software project communication because project information is a byproduct of the development processes—this is a major design feature of adaptive life cycles. However, reliance on face-to-face interaction

requires participation by the appropriate project stakeholders (customer, users, users' representatives, and others). Stakeholder attendance at planning meetings and iterative demonstrations of the evolving product is crucial. Where face-to-face communications are not possible, more traditional communications will be required.

Also, ongoing engagement of and communication with stakeholders is important throughout the entire project life cycle, because requirements, assumptions, and constraints often change as a software project evolves. And, it is important to ensure that project stakeholders receive the information they need during planning meetings, product demonstrations, and project retrospectives. Stakeholders should be encouraged to actively participate in these meetings. They should be asked what information they need, and every attempt should be made to provide it.

10.2 Manage Software Project Communications

The inputs, tools and techniques, and outputs in Section 10.2 of the *PMBOK® Guide* are applicable to managing software project communications.

10.2.1 Manage Software Project Communications: Inputs

The inputs in Section 10.2.1 of the *PMBOK® Guide*, are applicable inputs for managing software project communications. In addition, the input in 10.2.1.5 applies to managing software project communications.

10.2.1.1 Communications Management Plan

See Section 10.2.1.1 of the *PMBOK® Guide*.

10.2.1.2 Work Performance Reports

See Section 10.2.1.2 of the *PMBOK® Guide*.

10.2.1.3 Enterprise Environmental Factors

See Section 10.2.1.3 of the *PMBOK® Guide*.

10.2.1.4 Organizational Process Assets

See Section 10.2.1.4 of the *PMBOK® Guide*.

10.2.1.5 Release and Iteration Plans

As stated in Section 10.1.1 of this software extension, adaptive life cycles for software projects often include iteration and release plans. These plans provide an important input for the managing software project communications.

10.2.2 Manage Software Project Communications: Tools and Techniques

Because of the potential for high rates of change and the lack of a tangible, evolving product, managing communications on software projects is especially important. Project information can be provided via push and pull mechanisms. Information such as status reports should be pushed out to stakeholders on a regular basis (perhaps weekly). Information can be published in a repository so that stakeholders can access the desired information at the desired level of detail.

The tools and techniques in Section 10.2.2 of the *PMBOK® Guide* are applicable tools and techniques for managing software project communications. In addition, Section 10.2.2.6 –10.2.2.9 are applicable to managing software project communications.

10.2.2.1 Communication Technology

See Section 10.2.2.1 of the *PMBOK® Guide*.

10.2.2.2 Communication Models

See Section 10.2.2.2 of the *PMBOK® Guide*.

10.2.2.3 Communication Methods

See Section 10.2.2.3 of the *PMBOK® Guide*.

10.2.2.4 Information Management Systems

See Section 10.2.2.4 of the *PMBOK® Guide*.

10.2.2.5 Performance Reporting

See Section 10.2.2.5 of the *PMBOK® Guide*.

10.2.2.6 Information Radiators

As stated previously, information radiators are large, graphic displays of the project status and metrics that can be used to distribute software project information. They are frequently updated and located where the project team can easily see them. Common kinds of information radiator graphs include task boards, burn-up and burn-down graphs, defect reports, status of rework, and so forth.

A storyboard is a kind of information radiator for a software project. Post-it or similar sticky notes are placed on a white board to describe project tasks. The columns of a storyboard can be used to display items such as stories, all tasks, tasks in progress, tasks completed, story bugs (defects), and tasks completed. The rows show the progress of the items in columns. A storyboard is depicted in Figure 10-2.

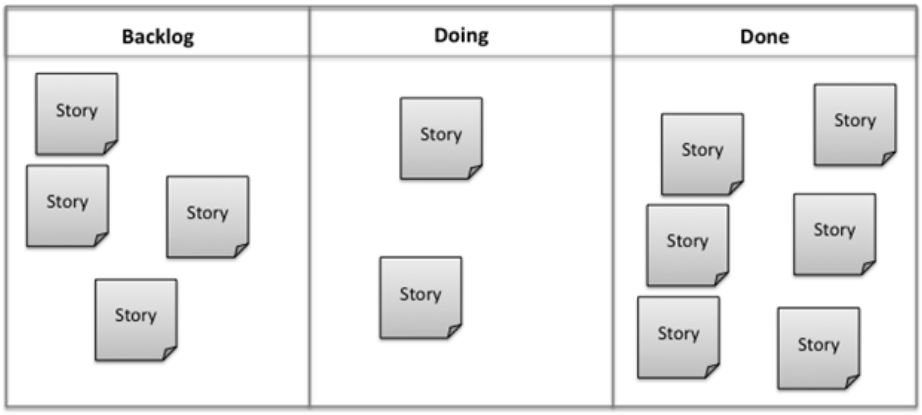


Figure 10-2. Depiction of a Storyboard

10.2.2.7 Velocity Statistics

Software projects that use predictive life cycles track velocity data on each iteration of the development cycle and communicate it to the project team. Velocity is a measure of the work completed by the project team during each iteration. It is an indicator of team capacity and a measure of productivity and progress. Velocity can be tracked using person-day metrics (i.e., ideal developer-days) or in less precise measures of time, such as story points completed.

10.2.2.8 Yesterday's Weather

"Yesterday's weather" describes the team's productivity during the previous period (i.e., velocity). It is useful because it reflects the fully loaded capability of the team's resources including the impacts of defect remediation, support, and other work demands. Using yesterday's weather is a reliable way of estimating the team's current iteration capacity.

Yesterday's weather uses recent performance as an indicator for likely future performance. For example, if the team completed 30 story points last week, using 30 story points as an estimate for progress this week is probably more valid than using the 45 story points per week estimated at the beginning of the project.

10.2.2.9 Online Collaboration Tools

Online collaboration tools can also be used so that those who are remote from the team can visit the project website and view project information such as big visible charts and yesterday's weather.

10.2.3 Manage Software Project Communications: Outputs

The outputs in Section 10.2.3 of the *PMBOK® Guide* are applicable outputs for managing software project communications. In addition, the outputs in 10.2.3.5 – 10.2.3.7 are applicable to managing software project communications.

10.2.3.1 Project Communications

See Section 10.2.3.1 of the *PMBOK® Guide*.

10.2.3.2 Project Management Plan Updates

See Section 10.2.3.2 of the *PMBOK® Guide*.

10.2.3.3 Project Documents Updates

See Section 10.2.3.3 of the *PMBOK® Guide*.

10.2.3.4 Organizational Process Assets Updates

See Section 10.2.3.4 of the *PMBOK® Guide*.

10.2.3.5 Special Communication Tools

Software projects that use adaptive life cycles often use special communications tools to specify and measure scope, schedule, budget, progress, and risks. These communications may include product backlogs, release maps, cumulative flow diagrams, and risk burn-down charts. These terms are defined in the Glossary.

10.2.3.6 Online Collaboration Tools

Software projects often use online collaboration tools to share and communicate project status. These tools allow geographically dispersed members to access project information. These tools are also continuously available to a project group that may be located in a different time zone. Online collaboration tools can provide a rich environment for storing documents, images, videos of product demonstrations, and threaded discussion forums.

10.2.3.7 Updated Information Radiators

The project's information radiators are updated to reflect the latest available information. A burn-down chart and a parking lot diagram are illustrated in Figures 10-3 and 10-4. A cumulative-flow diagram is illustrated in Figure 6-5.

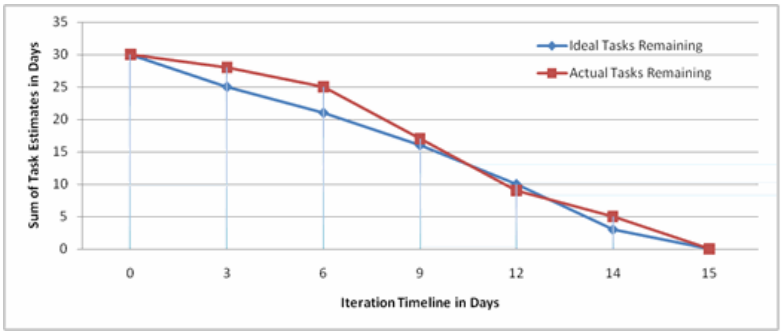


Figure 10-3. Burn-down Chart for a Software Project Iteration

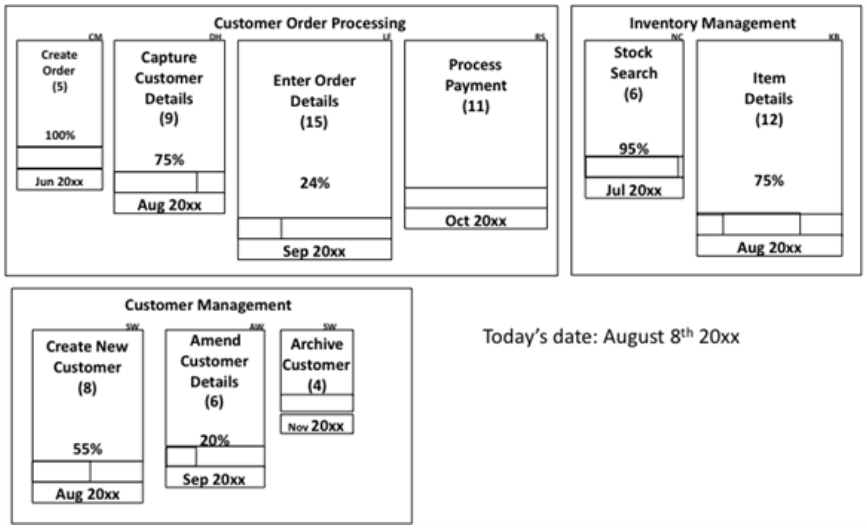


Figure 10-4. Parking Lot Diagram for a Software Project

10.3 Control Software Project Communications

According to Section 10.3 of the *PMBOK® Guide*, Control Communications is the process of monitoring and controlling communications throughout the entire project life cycle to ensure the information needs of project stakeholders are met. The techniques presented in Section 10.3 of the *PMBOK® Guide* are generally applicable to controlling communications in software projects. The following extensions to Section 10.3 are also applicable to

software projects.

Controlling communications for a software project involves providing insight into the

development progress as issues arise. There are many metrics that can be reported for software projects, but few are useful. Commonly reported and useful metrics include measures of cost, schedule, product volume, defects, and progress.

Good metrics are simple and relevant to the end goal of delivering an acceptable product within the constraints of schedule, budget, resources, technology, and other relevant factors. Software project measures should be byproducts of the processes used; it should not require an inordinate effort to produce them.

For adaptive life cycles, the content of the evolving product is the primary measure of progress. Iterations of the life cycle add increments to the evolving product. Newly added content, in combination with existing content, is tested and demonstrated at the end of the iterations. The demonstrations, in combination with the prioritized features in the product backlog (prioritized by business value), provide a measure of value-adding work that remains to be done. For adaptive life cycles, metrics such as stories developed (and tested) compared to stories remaining, meet the criteria of being simple to produce and relevant to the end goal.

Adaptive reporting tools such as cumulative flow diagrams, burn-up/burn-down graphs, and parking lot diagrams also provide valuable project information.

10.3.1 Control Software Project Communications: Inputs

The inputs in Section 10.3.1 of the *PMBOK® Guide* are applicable inputs for controlling software project communications. In addition, the inputs in 10.3.1.6 and 10.3.1.7 are applicable for controlling software project communications.

10.3.1.1 Project Management Plan

See Section 10.3.1.1 of the *PMBOK® Guide*.

10.3.1.2 Project Communications

See Section 10.3.1.2 of the *PMBOK® Guide*.

10.3.1.3 Issue Log

See Section 10.3.1.3 of the *PMBOK® Guide*.

10.3.1.4 Work Performance Data

See Section 10.3.1.4 of the *PMBOK® Guide*.

10.3.1.5 Organizational Process Assets

See Section 10.3.1.5 of the *PMBOK® Guide*.

10.3.1.6 Prioritized Backlog

For adaptive software project life cycles, the prioritized product backlog plays a key role in controlling communications. It is the primary method used to communicate the agreed-upon work and the sequence of upcoming development. The backlog can take the form of an online tool, a spreadsheet, or a stack of task cards.

10.3.1.7 Velocity Statistics and Projections

For adaptive life cycles, current velocity information and historical trends are used to determine the rate at which work was completed in previous iterations. This information is essential for estimating the amount of work that can be completed in subsequent iterations. Figure 10-5 illustrates a typical velocity chart for a software project.

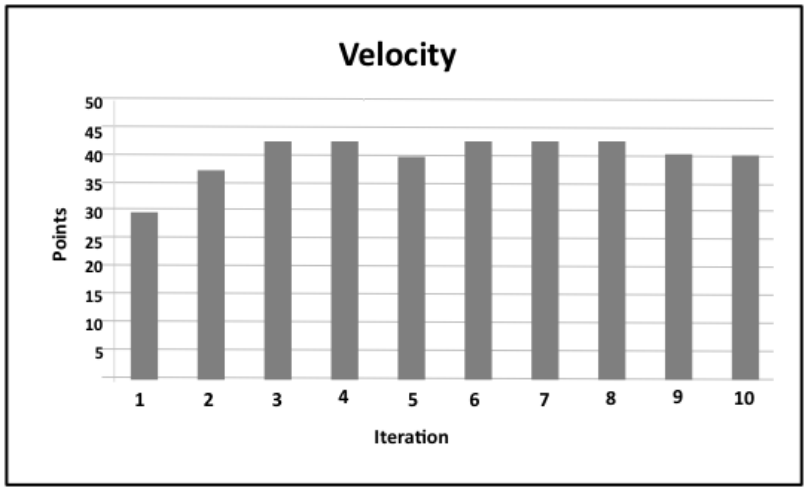


Figure 10-5. A Velocity Chart for a Software Project

10.3.2 Control Software Project Communications: Tools and Techniques

The tools and techniques in Section 10.3.2 of the *PMBOK® Guide* are applicable for managing software project communications. In addition, the tools and techniques in Sections 10.3.2.4 and 10.3.2.5 are applicable to managing software project communications.

10.3.2.1 Information Management Systems

See Section 10.3.2.1 of the *PMBOK® Guide*.

10.3.2.2 Expert Judgment

See Section 10.3.2.2 of the *PMBOK® Guide*.

10.3.2.3 Meetings

See Section 10.3.2.3 of the *PMBOK® Guide*.

10.3.2.4 Considerate Communications

Software development requires concentration in a quiet setting to enter the productive state of flow. It is often reported that this takes about 20 minutes to achieve and can be disrupted by a phone call or a minute or two of conversation **[36]**. This presents a paradox for software developers; on the one hand they need to get to a state of flow, but on the other hand, they need a co-located team environment with high bandwidth and face-to-face communications for resolving issues and obtaining rapid feedback.

One approach is, when possible, to arrange a work area that includes quiet rooms in which to work and a common work area where team members can discuss issues. Another approach is the use of quiet hours that provide the quiet atmosphere of a library. During specified quiet hours, phones are disabled and no visitors or meetings are scheduled.

The use of electronic messaging when team members are physically co-located in a common area can also be used to minimize the impact on concentration flow while still allowing communication. Regardless of the approach used, communications control should support the concentrated effort required for creative work.

10.3.2.5 Automated Systems

Systems that automatically collect project status can be used to improve communications efficiencies. These systems, often used to control communication in software projects include Wiki sites, project websites, and collaboration-based internet sites.

10.3.3 Control Software Project Communications: Outputs

The outputs in Section 10.3.3 of the *PMBOK® Guide* are applicable for controlling software project communications. In addition, the outputs listed in Sections 10.3.3.6 and 10.3.3.7 are also applicable.

10.3.3.1 Work Performance Information

See Section 10.3.3.1 of the *PMBOK® Guide*.

10.3.3.2 Change Requests

See Section 10.3.3.2 of the *PMBOK® Guide*.

10.3.3.3 Project Management Plan Updates

See Section 10.3.3.3 of the *PMBOK® Guide*.

10.3.3.4 Project Documents Updates

See Section 10.3.3.4 of the *PMBOK® Guide*.

10.3.3.5 Organizational Process Assets Updates

See Section 10.3.3.5 of the *PMBOK® Guide*.

10.3.3.6 Release/Iteration Plan Updates

An important part of managing stakeholder expectations is ensuring there are no surprises for anyone. So, for adaptive life cycles, it is important to distribute and explain updates to release and iteration plans that are typically amended frequently. Release plans describe when releases will occur and what functionality they will contain. Iteration plans confirm the work the project team has committed to deliver at the end of the next iteration. An iteration plan provides an early indication of the functionality that will be user acceptance-tested and demonstrated at the end of each iteration.

10.3.3.7 Reprioritized Backlog

When using adaptive life cycles for software projects, the customer has the opportunity to reprioritize the backlog of features to be developed throughout the project lifecycle. The backlog of work describes the remaining work to be done and the current priorities. The work backlog also shows the planned sequence of development and, using average velocity and "yesterday's weather," the schedule for development of these features can be estimated. The backlogs of product features and remaining work, along with estimates of scheduled feature delivery, are important elements of project communication that help to control customer/product owner expectations and eliminate unpleasant surprises.

11 Software Project Risk Management

According to Section 11 of the *PMBOK® Guide*, Project Risk Management includes the processes of conducting risk management planning, identification, analysis, response planning, and controlling risk on a project. The objectives of Project Risk Management are to increase the probability and impact of positive events, and decrease the probability and impact of negative events in the project. This section of the *Software Extension to the PMBOK® Guide – Fifth Edition* addresses risk management for software projects by describing risks and risk mitigation strategies that are important for managing software projects, and which merit attention beyond that provided in the *PMBOK® Guide.*

As defined in the Glossary, risk is an uncertain event or condition that, if it occurs, has a positive or negative effect on a project's objectives. In ISO Guide 73:2009 – Risk Management: Vocabulary **[40]**, risk is defined as the "combination of the probability of an event and its consequence." This widely used definition is applied in the principal software engineering standard for risk management: ISO/IEC/IEEE 16085 – Systems and

software engineering—Life cycle processes—Risk management **[41]**.

Each software development project has different uncertainties and risks, because each project is a unique combination of requirements, design, and construction, resulting in distinct software products (uncertainty arises from a lack of information; risk is a potential issue). Software technical risks and software project risks affect every stakeholder. Therefore, almost every recommended activity in the *PMBOK® Guide* and this software extension is intended to manage risks. Software risk management aims to improve the probability of achieving the project goals; software opportunity management aims to exceed the project goals. Opportunity management is commonly applied in software project management, especially in adaptive projects that have the opportunity to respond to customer-requested changes, apply new technology, or receive additional resources. The risk management process is "a continuous process for systematically identifying, analyzing, treating, and monitoring risk throughout the life cycle of a product or service" **[41]**.

Software project risk management and opportunity management for software projects includes planning, identifying, and analyzing software project risks and opportunities; performing software project qualitative and quantitative risk and opportunity analyses; planning risk and opportunity responses; and monitoring and controlling project risks and opportunities. Figure 11-1 provides an overview of the Project Risk Management processes. Commonly occurring risks for software projects include technical, schedule, cost, quality (e.g., security, safety, availability), team dynamics, and customer/stakeholder risk factors. Risk treatments include accepting, avoiding, transferring, or mitigating risk. Mitigating risk can occur by either immediate action or tracking and deferred action, if warranted.

While this section primarily addresses software development project risk management, the techniques and approaches are also applicable to delivery of software as a service. In that case, the primary risk is a break in service continuity, that is, the inability to continually deliver services at agreed-upon levels.

Figure 11-1 provides an overview of Software Project Risk Management; it is an adaptation of Figure 11-1 in the *PMBOK® Guide*.

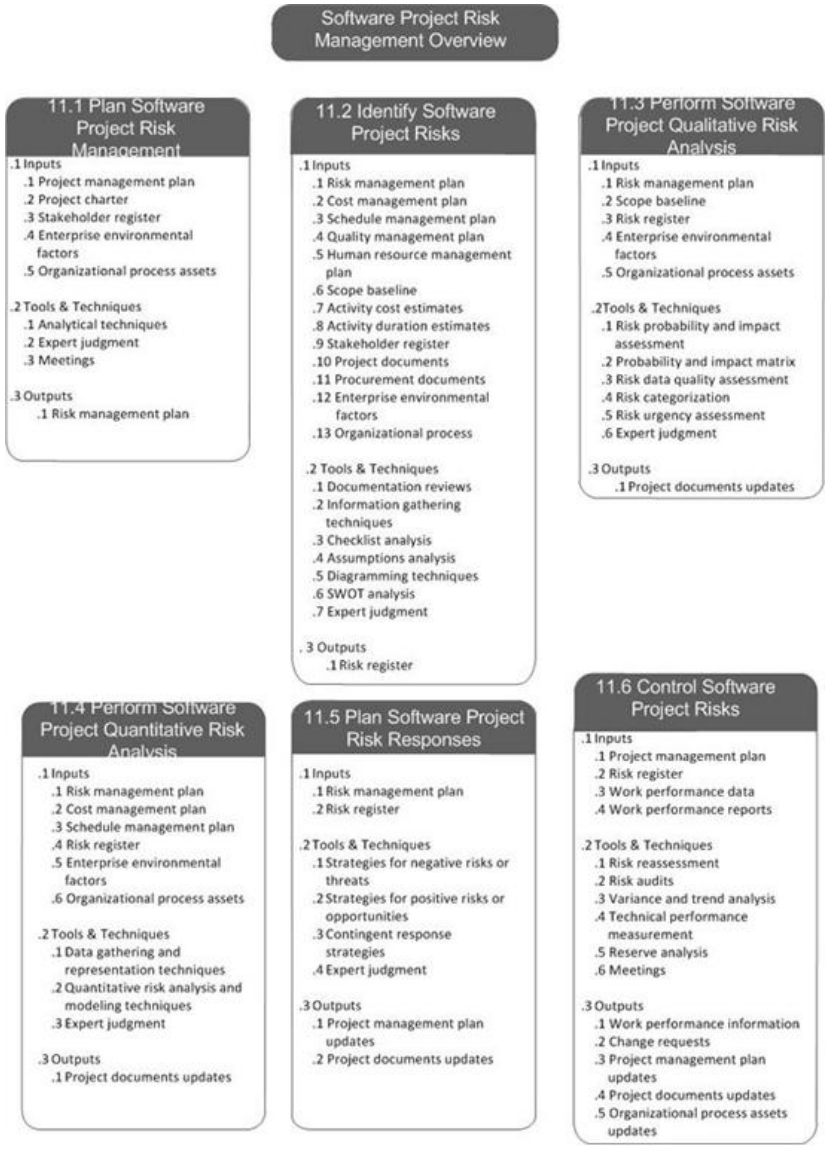


Figure 11-1. Software Project Risk Management Overview

11.1 Plan Software Project Risk Management

The inputs, tools and techniques, and outputs for planning risk management in Section 11.1 of the *PMBOK® Guide* are applicable to planning risk management for software projects with the following additions and clarifications.

11.1.1 Plan Software Project Risk Management: Inputs

The inputs for planning risk management in Section 11.1.1 of the *PMBOK® Guide* are applicable for planning software project risk management.

11.1.1.1 Project Management Plan

See Section 11.1.1.1 of the *PMBOK® Guide*.

11.1.1.2 Project Charter

See Section 11.1.1.2 of the *PMBOK® Guide*.

11.1.1.3 Stakeholder Register

See Section 11.1.1.3 of the *PMBOK® Guide*.

11.1.1.4 Enterprise Environmental Factors

See Section 11.1.1.4 of the *PMBOK® Guide*.

11.1.1.5 Organizational Process Assets

See Section 11.1.1.5 of the *PMBOK® Guide*.

The following considerations are also applicable. Software risk planning occurs repeatedly, beginning with an initial formal or informal risk-benefit analysis and decision either to initiate or not initiate the project. For large, formal software projects, projects in regulated environments, and projects involving safety-critical software, a documented risk management plan is essential. Most projects have less formal risk management procedures or follow an overall enterprise risk management plan. Specialized domain knowledge may make risks more apparent to some team members. While all team members should be responsible for identifying and communicating risks, there should be a risk management lead.

Risk management planning accompanies project planning and is reflected in the project at many levels, including risk management activities, data gathering, monitoring, decisions and assessments and changes to work plans. Depending upon the nature of risks, the life cycle model and processes are adjusted. Each of the assumptions and constraints used to develop the project plan should be examined for risk. Projects can take a proactive risk-driven approach. Prioritizing high-risk items and tackling them early in the project while there is time to try alternative approaches and to improve upon initial efforts accomplish this. By proactively undertaking high-risk work early, the software project team can reduce the overall impact to the project. By deferring risky work, problems may result and the probability of rework or a revised approach is much higher while the time remaining to recover from problems is short. Simply put, it is more cost-effective to resolve risks earlier than later.

11.1.2 Plan Software Project Risk Management: Tools and Techniques

The tools and techniques for planning risk management in the *PMBOK® Guide* are applicable tools and techniques for planning software project risk management.

11.1.2.1 Analytical Techniques

See Section 11.1.2.1 of the *PMBOK® Guide*.

11.1.2.2 Expert Judgment

See Section 11.1.2.2of the *PMBOK® Guide*.

11.1.2.3 Meetings

See Section 11.1.2.3 of the *PMBOK® Guide*.

The following considerations also apply. Adaptive life cycle software projects pull requirements and user stories from a backlog that can entail frequent reprioritization to permit risk management actions as early as possible in the life cycle, minimizing delayed and compounded effects. Also, since integration and regression testing is built into each iterative cycle, the probability of untested risky elements remaining in the product towards the end of the project is greatly reduced. Adaptive life cycles can be called risk-driven, because the software project team can pull high-risk stories forward from the backlog.

Adaptive life cycle projects allow for frequent reassessment of risks and reprioritization at the end of each iteration, which can take advantage of newly identified opportunities to add features or take action to mitigate newly identified risks. The project team can

add risk avoidance and risk reduction actions into the backlog and choose to proactively attack the risks before they have an impact on the project. The team should think of risk avoidance and risk mitigation as part of the value proposition for the adaptive planning cycle.

11.1.3 Plan Risk Management: Outputs

The output for planning risk management in Section 11.1.3 of the *PMBOK® Guide*, the risk register, is applicable to planning software project risk management.

11.1.3.1 Risk Management Plan

See Section 11.1.3 of the *PMBOK® Guide.*

In addition, when planning for the next iteration of an adaptive life cycle, the project team typically balances delivering business value with risk reduction. Sometimes the team may select a next feature to be implemented that has the best return on investment. Sometimes they will undertake an action to avoid or mitigate risk since the impact of the risk occurring would be greater than the ROI value of the next feature in the backlog, as depicted in Figure 11-2.

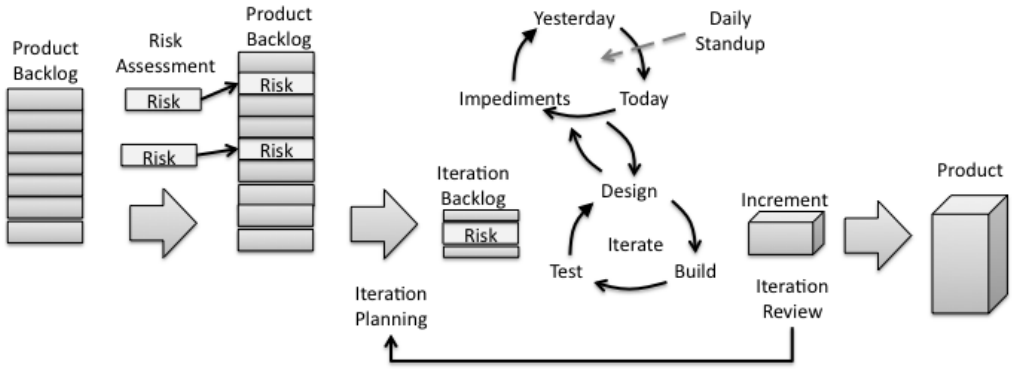


Figure 11-2. Business and Risk Reduction Activities Prioritized in the Product Backlog

The software project manager needs to ensure that risk management procedures, reporting rhythms, and risk register are established at the beginning of the project. The risk register can be as simple as a spreadsheet or whiteboard with annotated note cards or stories attached. For large projects and critical software products, specialized software tools aid in managing the outputs from planning risk management.

11.2 Identify Software Project Risks

The inputs, tools and techniques, and outputs for identifying risks in Section 11.2 of the *PMBOK® Guide* are applicable to identifying risks for software projects.

11.2.1 Identify Software Project Risks: Inputs

The inputs for identifying project risk in the *PMBOK® Guide* are applicable for identifying software project risks.

11.2.1.1 Risk Management Plan

See Section 11.2.1.1 of the *PMBOK® Guide*.

11.2.1.2 Cost Management Plan

See Section 11.2.1.2 of the *PMBOK® Guide*.

11.2.1.3 Schedule Management Plan

See Section 11.2.1.3 of the *PMBOK® Guide*.

11.2.1.4 Quality Management Plan

See Section 11.2.1.4 of the *PMBOK® Guide*.

11.2.1.5 Human Resource Management Plan

See Section 11.2.1.5 of the *PMBOK® Guide*.

11.2.1.6 Scope Baseline

See Section 11.2.1.6 of the *PMBOK® Guide*.

11.2.1.7 Activity Cost Estimates

See Section 11.2.1.7 of the *PMBOK® Guide*.

11.2.1.8 Activity Duration Estimates

See Section 11.2.1.8 of the *PMBOK® Guide*.

11.2.1.9 Stakeholder Register

See Section 11.2.1.9 of the *PMBOK® Guide*.

11.2.1.10 Project Documents

See Section 11.2.1.10 of the *PMBOK® Guide*.

11.2.1.11 Procurement Documents

See Section 11.2.1.11 of the *PMBOK® Guide*.

11.2.1.12 Enterprise Environmental Factors

See Section 11.2.1.12 of the *PMBOK® Guide*.

11.2.1.13 Organizational Process Assets

See Section 11.2.1.13 of the *PMBOK® Guide*.

The following considerations also apply. While every software project needs to identify risk factors, it is helpful to be aware of the more common types of risks for software

projects. The Software Engineering Institute (SEI) has published several three-level risk taxonomies, notably for operational risk and development project risk. These taxonomies break down risks by class, for example, program constraints, product engineering, and development environment; then by element, such as requirements within product engineering; and then by attribute, such as stability or formality of requirements.6

Table 11-1 is a simple first-level risk breakdown structure, with examples of common software project risks.

Table 11-1 First-Level Risk Breakdown Structure

Project Risk	Description
Technical	Software does not work as required: excessive defects; software does not scale to capacity or performance requirements; undefined or misunderstood requirements; late integration of software modules reveals errors during late testing; software does not fulfill customer expectations and needs; software is not easily usable by end users; excessive rework or refactoring due to unstable requirements, requirements inflation, or changes in scenarios; choice of new development platforms, languages, or tools with limited staff availability, software becomes corrupted due to inadequate configuration management of baseline, development work, and test versions; technology changes and upgrades during the project; external dependency on another project's ability to deliver usable, timely input.
Safety	Developed system has defects that can cause injury, death, or environmental destruction.
Security	Developed system integrity inconsistent with required software criticality (likelihood of severe consequences from malfunction); developers unfamiliar with probable security threats to the software; inadequate system design for access control, protection of personal or proprietary data at rest and in transit, and defense of system against malware and hacking; reuse of code with undetermined pedigree; disaster or security breach affects the development or production infrastructure.
Team	Inexperienced in the tools, organizational processes, development method, or customer business requirements; understaffed (staff not yet on board or pulled for other projects); staff burnout; staff turnover; communication and coordination issues within the team or with stakeholders, due to dispersed or virtual team or cultural differences; new staff pulling attention of experienced staff; multiple developers working on the same code branch.
Schedule	Baseline schedule is inconsistent with actual velocity; project won't finish essential or required features on time for scheduled release; scope creep impacts completion of original goals; delays in development lead to pressure to abbreviate testing; project completion measurements are not reflective of effective status (relying on SLOC or percent complete estimates); plans don't address initial architecture and data design or documentation or integration testing; test schedule allows time for only one run, ignoring the probability of retest.
Costs	Inaccurate estimates of labor rates and productivity/velocity, actual costs beyond available funding, unable to meet affordability challenge
Customer and stakeholders	Unavailability of business process data, unavailability of technical data on systems being replaced or interfaced, unavailability of acceptance criteria (or market needs analysis), unavailability of customer or user representatives for requirements/feature prioritization, user testing, and system acceptance.

11.2.2 Identify Software Project Risks: Tools and Techniques

The tools and techniques in Section 11.2.2 of the *PMBOK® Guide* for identifying risks are applicable to identifying risks for software projects.

11.2.2.1 Documentation Reviews

See Section 11.2.2.1 of the *PMBOK® Guide*.

11.2.2.2 Information Gathering Techniques

See Section 11.2.2.2 of the *PMBOK® Guide*.

11.2.2.3 Checklist Analysis

See Section 11.2.2.3 of the *PMBOK® Guide*.

11.2.2.4 Assumptions Analysis

See Section 11.2.2.4 of the *PMBOK® Guide*.

11.2.2.5 Diagramming Techniques

See Section 11.2.2.5 of the *PMBOK® Guide*.

11.2.2.6 SWOT Analysis

See Section 11.2.2.6 of the *PMBOK® Guide*.

11.2.2.7 Expert Judgment

See Section 11.2.2.7 of the *PMBOK® Guide*.

In addition, the following considerations also apply. During retrospective meetings, project teams evaluate the evolving system, review areas that may be behind in development, and discuss areas with the most problems and concerns regarding the remaining work to be done. In doing so, project risks may be uncovered.

Risks are often expressed as potential problems, but it is more effective to identify them as areas for improvement (i.e., opportunities). Software project risks may be expressed as if–then statements: "If [risk/opportunity becomes an actual issue], then [adverse/favorable outcomes will occur]." This form of risk statement focuses attention on the potential outcome, which is a necessary part of the probability × impact calculation in classic risk analysis. It also draws attention to the threshold trigger that would cause a potential problem (i.e., risk) to become a real problem.

11.2.3 Identify Software Project Risks: Outputs

The output in Section 11.2.3 of the *PMBOK® Guide* on identifying risks is applicable for identifying risks for software projects.

11.2.3.1 Risk Register

See Section 11.2.3.1 of the *PMBOK® Guide*.

11.3 Perform Software Project Qualitative Risk Analysis

The inputs, tools and techniques, and outputs in Section 11.3 of the *PMBOK® Guide* for performing qualitative risk analysis are applicable to performing qualitative risk analysis for software projects, with the following additions and clarifications.

It is human nature to focus on the more immediate risks, but advanced risk management is used to control the longer-term risks. Software product development needs to be sustainable, from various viewpoints: financial, team continuity, the software design framework, and the quality of code for future changes. Risk analysis should also look for immediate and sustained opportunities.

11.3.1 Perform Software Project Qualitative Risk Analysis: Inputs

The inputs for performing qualitative risk analysis in Section 11.3.1 of the *PMBOK® Guide* are applicable to performing qualitative risk analysis for software projects.

11.3.1.1 Risk Management Plan

See Section 11.3.1.1 of the *PMBOK® Guide*.

11.3.1.2 Scope Baseline

See Section 11.3.1.2 of the *PMBOK® Guide*.

11.3.1.3 Risk Register

See Section 11.3.1.3 of the *PMBOK® Guide*.

11.3.1.4 Enterprise Environmental Factors

See Section 11.3.1.4 of the *PMBOK® Guide*.

11.3.1.5 Organizational Process Assets

See Section 11.3.1.5 of the *PMBOK® Guide*.

In addition, considerations for performing qualitative risk analysis for software projects include: (1) criticality of the software product (its impact on its users and the operational environment), (2) the effect should a risk interfere with the completion and successful delivery of the software product, and (3) the overall effect on the producing organization (that is, a “bet the company” project or an optional enhancement to a mature product).

11.3.2 Perform Software Project Qualitative Risk Analysis: Tools and Techniques

The tools and techniques for performing qualitative risk analysis in Section 11.3.2 of the *PMBOK® Guide* are applicable to performing qualitative risk analysis for software projects.

11.3.2.1 Risk Probability and Impact Assessment

See Section 11.3.2.1 of the *PMBOK® Guide*.

11.3.2.2 Probability and Impact Matrix

See Section 11.3.2.2 of the *PMBOK® Guide*.

11.3.2.3 Risk Data Quality Assessment

See Section 11.3.2.3 of the *PMBOK® Guide*.

11.3.2.4 Risk Categorization

See Section 11.3.2.4 of the *PMBOK® Guide*.

11.3.2.5 Risk Urgency Assessment

See Section 11.3.2.5 of the *PMBOK® Guide*.

11.3.2.6 Expert Judgment

See Section 11.3.2.6 of the *PMBOK® Guide*.

The following considerations also apply. Qualitative analyses of risk are, by definition, difficult or impossible to quantify and are usually based on subjective and limited experience. Accurately estimating the probability of a risk requires a large experience base of similar projects (similar in complexity, criticality, infrastructure and tools, team experience, and organizational process resources). In practice, only very large organizations are able to accumulate experience bases and, for competitive reasons, are reluctant to share it with external stakeholders and other organizations. Often, experience concerning the rate of work completion (i.e., velocity) is not available until the project is well underway when there is less time to exert corrective measures.

Additional causal analysis (e.g. asking "why" three times) can help identify root causes of identified risks. Qualitative risk analysis benefits from recent experience, (e.g., for similar infrastructure and team members accustomed to working together). The analysis may be distorted by the impact of recent work (i.e., the tendency to emphasize the most recent experience rather than the long-term average). A risk that became a problem on a previous project may be considered likely to occur in the next project. Conversely, the problems encountered in previous projects may be considered to be thoroughly nullified by lessons learned and mitigations applied, so that the probability of recurrence is considered to be minimal. However, the precautionary mitigations may impose extraordinary costs in monitoring and control, such as attempting exhaustive testing, scheduling a large number of project reviews and executive presentations, and imposing heavy documentation requirements, which in themselves create the risk of excessive cost and noncompetitive business processes.

Performing both Qualitative and Quantitative Risk Analyses involves analyzing identified project risks and prioritizing then to identify the highest-risk items, features, and/or stories.

Qualitative ratings of risks for software projects can be based on subjective values such as low, medium, high, or very high for both probability and potential impact, as illustrated in Table 11-2. A low-risk exposure might correspond to a small schedule delay or cost overrun or a minor quality issue; a medium value to a more significant value of a project or product parameter, a high value to a major issue; and a very high value to a potentially catastrophic situation.

Quantitative ratings can be based on numeric values, as illustrated in Table 11-3. The

entries in Table 11-3 are the products of the corresponding values of probability and normalized impacts; this product is called the risk exposure. A project manager or assigned risk manager may assign both an unmitigated risk exposure and a mitigated risk exposure, along with the cost of risk mitigation. Risk leverage factors (the difference between unmitigated and mitigated risk exposures divided by the cost of risk mitigation) can be used to evaluate the effectiveness of various risk reduction strategies. See also Figure 11-11 in the *PMBOK® Guide*.

Table 11-2. A Qualitative Risk Exposure Matrix

Impact	Low	Medium	High	Very High
Probability				
Low	Low	Medium	High	Medium
Medium	Low	High	High	High
High	Medium	High	Very high	Very high
Very High	Medium	High	Very high	Extreme

Table 11-3. A Quantitative Risk Exposure Matrix

Impact	25	50	75	100
Probability				
0.25	6.25	12.5	18.75	25
0.5	12.5	25	37.5	50
0.75	18.75	37.5	56.25	75
0.95	23.75	47.5	71.25	95

For adaptive life cycle projects, a risk exposure matrix can be used to prioritize features for inclusion in the next iterative cycle by focusing on the features that will have the largest risk/return value for the business or the end users, as illustrated in Figure 11-2. This is similar to opportunity analysis, stated in risk management terms.

11.3.3 Perform Software Project Qualitative Risk Analysis: Outputs

The output for performing qualitative risk analysis in Section 11.3.3 of the *PMBOK® Guide* is applicable to performing qualitative risk analysis for software projects.

11.3.3.1 Project Documents Updates

See Section 11.3.3.1 of the *PMBOK® Guide*.

11.4 Perform Software Project Quantitative Risk Analysis

The inputs, tools and techniques, and outputs for performing quantitative risk analysis in Section 11.4 of the *PMBOK® Guide* are applicable to performing quantitative risk analysis for software projects.

Quantitative techniques are often used on major software projects, such as competitive software acquisitions or enterprise initiatives. The time and expertise required for extensive quantitative modeling may not be justified for shorter, simpler projects.

Quantitative analysis may be used to prioritize (1) unmitigated risks in the product backlog, and (2) risk avoidance and risk mitigation activities. A software project

technical risk has an impact cost, and a risk mitigation or risk transfer option has a quantifiable cost, such as the cost of procuring software in comparison to the labor cost of building the software—these can be translated into monetary units. Similarly, human resource and business risks can be estimated in monetary terms. Of course, not all risks have avoidance or mitigation steps that can be scheduled into a software project. Some risks may have to be accepted (e.g., the project is delayed while waiting for a procured component), but those steps that can be proactively addressed can be prioritized in the iterative project's backlog.

Quantitative risk analysis has several practical limitations. It is not possible to estimate the probability and impact of all potential problems (risks). For example, consider the risk of developing software that makes it easy for hackers to access private user data. There is no cost until after the project is ended when the software is being used in the operational environment. At that time, a security breach could result in fines, legal fees, and remediation costs to the users for credit monitoring, litigation costs, and loss of future business. The expenses are potentially large and serious, but not easily quantified at the time of risk analysis during software development.

For software projects, risk identification and risk analysis attempt to focus on the most probable and highest-impact risks rather than the cumulative impact of a succession of minor risks. Also, the impact of some risks may be hard to quantify as far as direct costs to the project or organization. The precision of the monetary value may not be great, but the point of quantitative risk analysis for software projects is usually to take action based on relative scorings rather than precise numbers. The goal is to reach consensus with the software project stakeholders on justifiable numbers to use as a basis for prioritization, not to report costs on a balance sheet.

The objectivity and relevance of a quantitative risk analysis ultimately depends on qualitative judgment, the availability of an experience base, and the objectivity of the experts estimating the best, most likely and worst-case points.

11.4.1 Perform Software Project Quantitative Risk Analysis: Inputs

The inputs in Section 11.4.1 of the *PMBOK® Guide* for performing quantitative risk analysis are applicable to performing quantitative risk analysis for software projects.

11.4.1.1 Risk Management Plan

See Section 11.4.1.1 of the *PMBOK® Guide*.

11.4.1.2 Cost Management Plan

See Section 11.4.1.2 of the *PMBOK® Guide*.

11.4.1.3 Schedule Management Plan

See Section 11.4.1.3 of the *PMBOK® Guide*.

11.4.1.4 Risk Register

See Section 11.4.1.4 of the *PMBOK® Guide*.

11.4.1.5 Enterprise Environmental Factors

See Section 11.4.1.5 of the *PMBOK® Guide*.

11.4.1.6 Organizational Process Assets

See Section 11.4.1.6 of the *PMBOK® Guide*.

11.4.2 Perform Software Project Quantitative Risk Analysis: Tools and Techniques

The tools and techniques in Section 11.4.2 of the *PMBOK® Guide* for performing quantitative risk analysis are applicable to performing quantitative risk analysis for software projects with the following extensions to Data Gathering and Representation Techniques and Quantitative Risk Analysis and Modeling Techniques.

11.4.2.1 Data Gathering and Representation Techniques

See Section 11.4.2.1 of the *PMBOK® Guide*.

The calculation of risk impact × probability is also used to calculate risk expected monetary value expressed as mitigation cost times probability.

Risk expected monetary value = Risk impact (in monetary units) × Risk probability (0.0 – 1.0)

A software project thus has a quantitative value (cost or benefit) for each risk mitigation activity in comparison to the cost and value of new work. For example, assume there is risk of having an inadequate reporting tool in place (assume $0 future cost for the existing tool) which could be completely mitigated by buying and using a high-performance reporting engine that costs $10,000 to buy, implement, and run. If the project estimates that there is a 50% chance of needing this tool, then the evaluated cost (or economic value) of purchasing the new tool is $5,000 ($10,000 × 50%).

On the other hand, even though the existing tool has a zero cost for use, suppose that the cost of staff time over the same period to collect and analyze data and compile reports is estimated to be $25,000 and, based on experience, there is again a 50% chance that reports will not be usable or ready on time. The cost of continuing with the existing tool is $12,500; therefore, purchasing the new tool is the better value.

Using this approach, a software risk manager can rank project risks to produce a prioritized list of risks ordered by expected monetary value, which can be used to prioritize the value of requirements in terms of risk. These activities actions support meaningful discussions with the software project sponsors. In Figure 11-3, the second risk from the top has an expected monetary value of $8000 × 50% or $4000. Therefore, when it comes to selecting requirements (features) for an upcoming iteration, the risk mitigation action associated with risk #2 ranks similar to the functional requirements value for number #2. In other words, this risk mitigation work is of equal value to the organization as the addition of a new software feature.

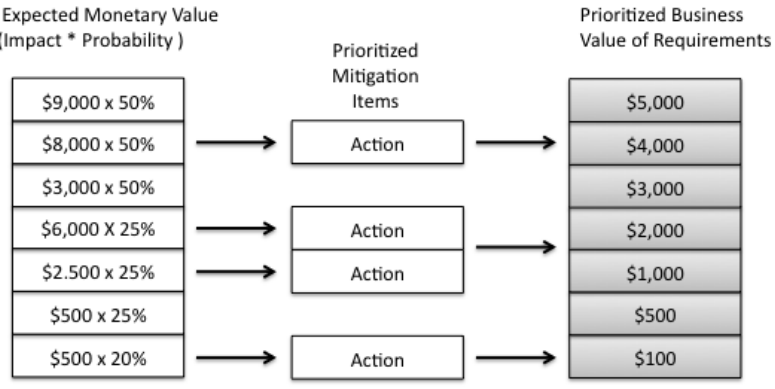


Figure 11-3. Comparative Priorities of Risk Treatment and Business Value

11.4.2.2 Quantitative Risk Analysis and Modeling Techniques

See Section 11.4.2.2 of the *PMBOK® Guide*.

The following consideration also applies. Quantitative simulations may identify the need for more extensive risk mitigation, and for applying schedule and budget contingency reserves. Monte Carlo simulations may be used to compute project outcomes at various levels of probability and to indicate the probability of obtaining the "most likely" estimate (see Figures 11-14 and 11-18 of the *PMBOK® Guide*).

11.4.2.3 Expert Judgment

See Section 11.4.2.3 of the *PMBOK® Guide*.

11.4.3 Perform Software Project Quantitative Risk Analysis: Outputs

The output in Section 11.4.3 of the *PMBOK® Guide* for performing quantitative risk analysis is applicable to performing quantitative risk analysis for software projects.

11.4.3.1 Project Documents Updates

See Section 11.4.3.1 of the *PMBOK® Guide*.

11.5 Plan Software Project Risk Responses

Planning risk responses for software projects includes the evaluation and selection of risk treatment alternatives. The project leaders evaluate the risk exposure of an untreated risk, the exposure after treatment (the *residual risk*), and the cost of the risk treatment. If the cost of the risk treatment is high compared to the impact of the risk, accepting the risk may be the best response. Accepted risks remain on a watch list or in a risk register for ongoing monitoring.

The inputs, tools and techniques, and outputs for planning risk responses in Section 11.5 of the *PMBOK® Guide* are applicable to planning risk responses for software projects.

11.5.1 Plan Software Project Risk Responses: Inputs

The inputs for planning risk responses in Section 11.5.1 of the *PMBOK® Guide* are

applicable to planning risk responses for software projects.

11.5.1.1 Risk Management Plan

See Section 11.5.1.1 of the *PMBOK® Guide*.

11.5.1.2 Risk Register

See Section 11.5.1.2 of the *PMBOK® Guide*.

11.5.2 Plan Software Project Risk Responses: Tools and Techniques

The tools and techniques for planning risk responses in Section 11.5.2 of the *PMBOK® Guide* are applicable for software projects.

11.5.2.1 Strategies for Negative Risks or Threats

See Section 11.5.2.1 of the *PMBOK® Guide*.

11.5.2.2 Strategies for Positive Risks or Opportunities

See Section 11.5.2.2 of the *PMBOK® Guide*.

11.5.2.3 Contingent Response Strategies

See Section 11.5.2.3 of the *PMBOK® Guide*.

11.5.2.4 Expert Judgment

See Section 11.5.2.4 of the *PMBOK® Guide*.

In addition to the tools and techniques for planning risk responses in Section 11.5.2 of the *PMBOK® Guide,* risk-based testing assesses the probability of elements of the software being defective and the consequences of those defects. Where the probability and consequence are high, that element is tested more extensively. Where the probability and consequence are low, that element is either tested less extensively or not tested at all. This allows the limited testing resources on a software project to be focused on providing the highest benefit.

Risk leverage factors (RLF) may be calculated for various risks as an aid to planning software project risk responses by calculating the risk exposure of an untreated risk (REut = probability × impact), risk exposure after treatment (REat = residual probability × impact) and including the cost of the risk treatment, RTc, where all three factors are expressed in monetary terms:

RLF = [REut – REat]/RTc

Risk leverage factors for identified risks can be used to prioritize the application of limited risk treatment funds. Higher values of RLF indicate higher cost-return benefits.

11.5.3 Plan Software Project Risk Responses: Outputs

The outputs for planning risk responses in Section11.5.3 of the *PMBOK® Guide* are applicable for software projects.

11.5.3.1 Project Management Plan Updates

See Section 11.5.3.1 of the *PMBOK® Guide*.

11.5.3.2 Project Documents Updates

See Section 11.5.3.2 of the *PMBOK® Guide*.

The following considerations also apply. Aside from quickly remedied risks, risk responses to avoid, transfer, or mitigate risk may require detailed planning to execute. The summary maintained in the risk register should include the specific risk treatment planned, who is responsible, affected stakeholders, cost and schedule for the risk treatment, monitoring

schedule, and measures to evaluate the progress and effectiveness of the risk treatment. The impact of the risk treatment should also be noted; the risk treatment may itself introduce secondary risks, safety concerns, or environmental impacts.

Table 11-4 contains typical risk responses used to avoid, mitigate, or transfer risk for software projects. No special approaches need to be stated for accepting risk; often the risk is accepted until a more cost-effective way to mitigate or transfer the risk is identified.

Table 11-4 Typical Risk Responses for Software Projects

Risk Response	Description
Technical	*Avoid Risk:* Use proven development platform and language. Change the requirements. *Transfer:* Use commercially available tools and modules or reuse existing software modules rather than creating new designs (buy rather than build). *Mitigate:* Engage the constant involvement of customers and developers. Work in short iterations so that risk can be identified early and development for risk mitigation has time to make an impact. Train the team on new development methods; obtain project sponsor commitment to the changes. Conduct regression testing for changes to critical software that may impact downstream modules or overall performance.
Security	*Avoid:* While there is no way to avoid all security risks and threats, use secure coding and access control techniques, and accredited architectures, follow security standards. *Transfer:* Obtain software kits and tools from recognized sources with a commitment to remediate security vulnerabilities. Recognized sources include the open source community as well as proprietary commercial software vendors. *Mitigate:* Train developers in secure coding. Engage intrusion detection and independent software penetration testers for software certification.
Team	*Avoid Risk:* Use a dedicated, experienced manager and teams, and established organizational processes. *Transfer:* Use collaborative processes so there is no single point of failure; engage recruiting or contract labor providers to offer backup or surge staff. (Note that adding staff late in a project often slows the project further while new staff come up to speed.) *Mitigate:* Balance staff between more expensive senior staff and less costly junior resources with coaching and training. Improve team communication methods to avoid duplicative work or rework.
Schedule	*Avoid Risk:* Review baseline schedule for accuracy in proportionate allocation of time to activities, resource loading and critical path. Allow time for planning and design before beginning large-scale development. *Transfer:* Involve customers in change control decisions at project checkpoints or sprint priorities and content. Get the team involved in planning and estimating. *Mitigate:* Start critical and higher-risk activities early in the schedule to allow time to prototype, test, iterate, integrate, and retest. Build reserve into the schedule. Get early feedback on variance from schedule and adjust iterative plans.
Cost	*Avoid Risk:* Estimate by function points completed and tested, rather than by SLOC or percent complete estimates. Use multiple cost estimating techniques. *Transfer:* Offer change proposals to include the customer in the cost of unexpected issues or the benefit of cost-saving opportunities. *Mitigate:* Shift resources from less critical activities or de-scope lower priorities.
Customer and stakeholders	*Avoid Risk:* Develop a project charter, contract or work agreement to clarify roles and expected customer responsibilities. *Transfer:* Designate a customer representative to represent the voice of the user with multiple sponsoring organizations. *Mitigate:* Specify contingencies and assumptions in the absence of customer data. Conduct walkthroughs and prototypes to build customer acceptance.

11.6 Control Software Project Risks

The inputs, tools and techniques, and outputs for controlling risks in Section 11.6 of the *PMBOK® Guide* are applicable to controlling risks for software projects with the following additions and extensions.

Risk monitoring and control for software projects includes tracking identified risks,

monitoring residual risks, executing risk treatment plans, and evaluating their effectiveness. On small software projects, monitoring and controlling risks are part of the project manager's duties. On large programs, another individual, often a quality assurance or planning specialist, is designated as the risk manager and is delegated the responsibility for recording new risks in the risk register and ensuring that risk mitigations are being implemented and completed by agreed-upon completion dates.

11.6.1 Control Software Project Risks: Inputs

The software project manager (or risk manager) typically schedules regular reviews of risk impacts and probabilities until risks are closed out. Risk managers also capture experience data and lessons learned for use in future phases and other projects.

Organizations vary in their tolerance of risk. The *risk threshold* is the point at which a the probability of a risk becomes large enough that it can no longer be accepted and needs further treatment. To determine when the risk threshold is reached, software projects use indicators such as technical performance measures (TPM) or more selectively, key performance indicators (KPI), which show how successfully a risk is being managed. For example, if the churn in requirements exceeds a defined percentage, or the number of test defects per thousand lines of code (KLOC) passes a defined level, or the cost or schedule performance index (CPI or SPI) exceeds a pre-specified limit, a risk threshold has been reached. This condition is called a *risk trigger* for the risk manager to initiate a contingency plan for risk treatment.

The inputs for controlling risks in Section 11.6.1 of the *PMBOK® Guide* are applicable to controlling risks for software projects.

11.6.1.1 Project Management Plan

See Section 11.6.1.1 of the *PMBOK® Guide*.

11.6.1.2 Risk Register

See Section 11.6.1.2 of the *PMBOK® Guide*.

11.6.1.3 Work Performance Data

See Section 11.6.1.3 of the *PMBOK® Guide*.

11.6.1.4 Work Performance Reports

See Section 11.6.1.4 of the *PMBOK® Guide*.

11.6.2 Control Software Project Risks: Tools and Techniques

The tools and techniques for controlling risks in Section 11.6.2 of the *PMBOK® Guide* are applicable to planning risk responses for software projects.

11.6.2.1 Risk Reassessment

See Section 11.6.2.1 of the *PMBOK® Guide*.

11.6.2.2 Risk Audits

See Section 11.6.2.2 of the *PMBOK® Guide*.

11.6.2.3 Variance and Trend Analysis

See Section 11.6.2.3 of the *PMBOK® Guide*.

11.6.2.4 Technical Performance Measurement

See Section 11.6.2.4 of the *PMBOK® Guide*.

11.6.2.5 Reserve Analysis

See Section 11.6.2.5 of the *PMBOK® Guide*.

11.6.2.6 Meetings

See Section 11.6.2.6 of the *PMBOK® Guide*.

The following considerations also apply. Adaptive life cycle software projects incorporate many mechanisms for dealing with change (an easily reprioritized backlog, short iterations, daily stand-up meetings, frequent demonstrations of working, deliverable software, and planning retrospectives) that also lend themselves to proactive response to risks as follows:

- 0**Daily stand-up meetings.** At the daily stand-up meeting, asking if there are any issues or impediments blocking progress can surface new project risks from the development team as today's issues and blockers could become tomorrow's risks and problems for the project. So it is important to pay attention to the issues being raised and transfer any appropriate issues to the risk log and undertake the necessary risk assessment steps. Also, when the team reports "impediments to progress" at the daily stand-up meetings, these may be candidates for potential risks, so the risk management plan should account for this iterative nature of review and potential source of risk identification. From a risk management perspective, the purpose of the daily meeting is for the team to identify potential new risks, issues, or signs of trouble, which if left unchecked could be a real threat to the project. The daily meetings also overcome the risk that team members will not prioritize their time productively or will be less able to solve technical issues without coordination. Over the course of a project, adaptive software deveopment teams use tools such as risk burn-down graphs and risk profiles to illustrate the effectiveness of the risk-driven approach. The goal is to rapidly reduce risks on the project. During a retrospective meeting immediately following an iterative cycle a team may close out risks that have been eliminated, and reevaluate the likelihood of risks occurring or recurring during the next period.

- 0**Retrospective results**—Retrospectives that regularly review the project for work that went well in addition to work that did not go well are good vehicles for identifying risk for the project.

- 0**Software evaluation feedback**—Prototype evaluations reveal stakeholder concerns with the proposed solution, which can result in technical and schedule risks. Addressing the concerns will likely require updates to the release and iteration plans and reprioritization of risk mitigations.

Adaptive life cycles provide some techniques for good risk management; they do not risk-proof or insulate projects from risks. Indeed, if an adaptive approach is new to an organization, then its introduction will be a risk; anything new represents a risk of misapplication, misunderstanding, confusion and failure. Having an active project sponsor and informed stakeholder involvement, wisely selecting team leaders with experience using

the adaptive approach, and scheduling time for team training are common techniques to overcome the risk of introducing new methods and processes.

11.6.3 Control Software Project Risks for Software Projects: Outputs

The outputs in Section 11.6.3 of the *PMBOK® Guide* for controlling risks are applicable for controlling risks for software projects.

11.6.3.1 Work Performance Information

See Section 11.6.3.1 of the *PMBOK® Guide*.

11.6.3.2 Change Requests

See Section 11.6.3.2 of the *PMBOK® Guide*.

11.6.3.3 Project Management Plan Updates

See Section 11.6.3.3 of the *PMBOK® Guide*.

11.6.3.4 Project Documents Updates

See Section 11.6.3.4 of the *PMBOK® Guide*.

11.6.3.5 Organizational Process Assets Updates

See Section 11.6.3.5 of the *PMBOK® Guide*.

In addition, risk burn-down charts, similar to progress burn-down charts, can be used to track the number of risks identified and closed out; they can be displayed as visual charts for the project team. Information in a risk register can be summarized in a *risk profile.* According to ISO/IEC/IEEE Standard 16085 **[41]** the project risk profile includes the risk management context, a record of each risk's current and historical state and priority, and all of the risk action requests. The risk action state includes the probability, consequences, and risk threshold for the risk.

12 Software Project Procurement Management

The introduction to Section 12 of the *PMBOK® Guide* states: Project Procurement Management includes the processes necessary to purchase or acquire products, services, or results needed from outside the project team. The organization can be either the buyer or seller of the products, services, or results of a project.

Large software organizations, like other engineering organizations, typically have a procurement department that deals with contracting issues related to the procurement of products and services. Small software organizations may not have a similar support function and, as a result, the software project manager may play an increased role in managing software project procurements.

Also, as indicated in the introductory paragraph of Section 12 of the *PMBOK® Guide*, an organization may be the seller of the products, services, or results of a project. In some cases, a software organization may be a prime contractor or a subcontractor (seller) to another organization or governmental agency. In these cases, some or all of the processes

to be followed and the metrics to be reported by the software project manager may be elements of the statement of work for the project.

This section of the *Software Extension to the PMBOK® Guide – Fifth Edition* focuses on the considerations involved in procuring services for a software project or new software products, such as a procuring a custom-built software application or turn-key infrastructure. It addresses planning, conducting, controlling, and closing out software project procurements, primarily from the point of view of the acquiring software project manager. It also addresses the acquisition of commercially available (COTS) software for use on a software project. Licensing of software packages, obtaining rights to modify open source software, reuse of existing components, and the purchase of specialty services to build software are all elements of software procurement.

Software may also be procured as a service. Just as with commercially available software, it is important to understand the exact nature of the services provided; how they might evolve over time; and what control the customer retains over the data provided to be processed under the service, the results obtained, and any security obligations. These considerations are usually covered in a service level agreement (SLA). Often, the standard agreement issued by the provider may not meet the acquirer's (i.e., the software project's) specific needs.

Procured services can include complete outsourcing of software development, assistance from software consultants and experts in software development processes, staff augmentation by contracted developers and testers, and provision of supporting services such as data migration and conversion, QA, CM, and product documentation.

Because software requires frequent updates to meet changes in security threats, infrastructure upgrades, and new functional requirements, it is rarely purchased outright without provisions for ongoing maintenance. In some cases, the software acquirer will obtain a license or right to use the software or a lease with specific terms and conditions rather than having complete control of the software source code. If there is no initial purchase price, as with freeware or open source software, the refresh and adaptation costs, as well as support costs, versioning, and maintenance are procurement considerations.

This section does not address the specialized work of procurement agents such as subcontract administrators and software buyers, nor does it address the legal and regulatory particularities of contracts and agreements for software, documentation, and other intellectual property, which vary from country to country. Successful suppliers and vendors of software establish and use effective processes, work as part of a mature organization, and are likely to do the following:

- 0Plan and execute acquisitions predictably,
- 0Support those programs with a management infrastructure that will enable them to succeed,
- 0Communicate more accurately and frequently, and
- 0Stay on schedule and deliver acceptable quality products.

Figure 12-1 provides an overview of Software Project Procurement Management; it is an adaptation of Figure 12-1 in the *PMBOK® Guide*.

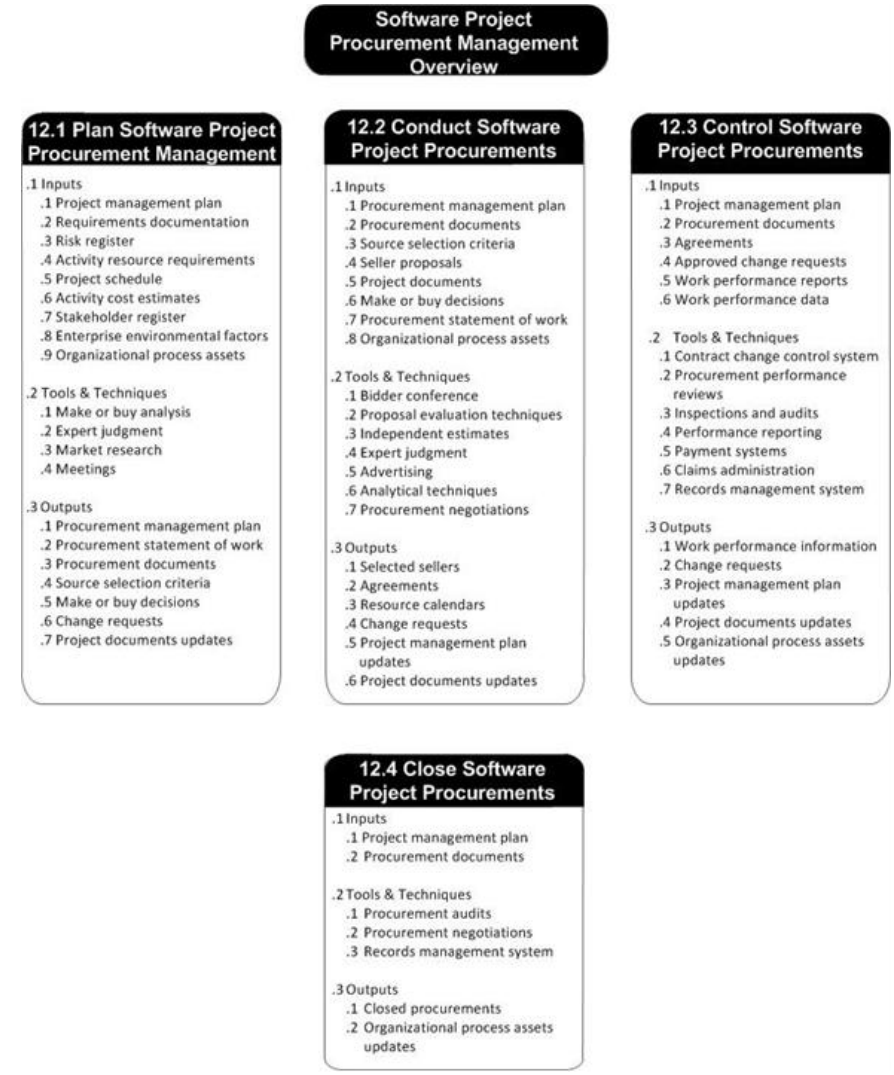


Figure 12-1. Software Procurement Management Overview

12.1 Plan Software Project Procurement Management

The first step in planning for software procurement is making the decision that a software product or service needs to be acquired. The organization may perform a business case analysis, trade study, or market survey of available capabilities, conduct a needs assessment, or a make-or-buy study to determine that the best way to meet a resource need is to acquire a software product or service (see Section 12.1.2.1 of the *PMBOK® Guide*). It is a good practice to document the alternatives considered before proceeding with procurement and to communicate the procurement strategy to the project stakeholders.

Once the decision to procure is made, the organization needs a procurement strategy. In fact, there may be a formal plan and schedule for the procurement. Major activities in planning the procurement of software include: (1) identifying potential suppliers, (2) specifying the requirements by preparing a statement of objectives or statement of work, (3) establishing evaluation criteria, (4) preparing preferred terms and conditions, and (5) developing procurement documents (request for proposals or request for tender).

• 0**Identifying suppliers**. Bidders' conferences with potential suppliers are often conducted as part of the initial market surveys. Architectural and technical decisions may severely limit the options for potential suppliers, since the supplier should have experience with the intended software environment. Procuring infrastructure such as an operating system, middleware, or a common development environment will drive how custom software capabilities are created. Conversely, the architecture of the software product needs to provide the necessary organization, infrastructure, and interfaces to enable integration of special functionality, application support, or utility software that may be procured.

• 0**Statement of objective or statement of work.** The choice of how to specify requirements for procurement depends on the scope, impact, and audience affected by the software or service, as indicated in Figure 12-2.

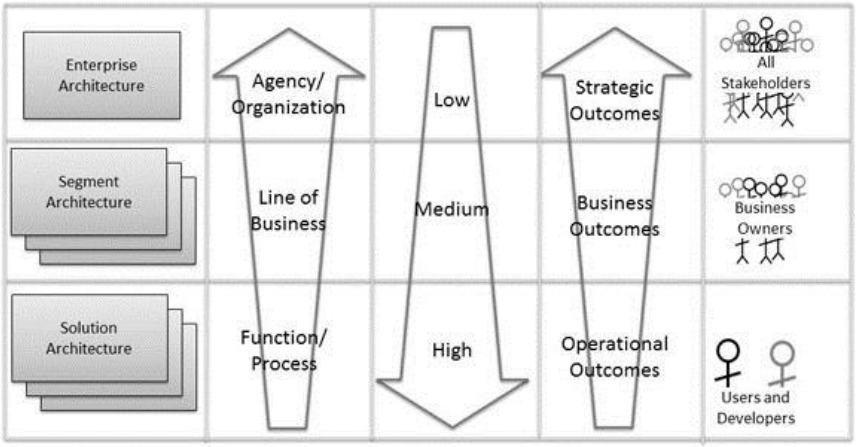


Figure 12-2. Level of Detail for Software/Service Acquisition Requirements

In all cases, the acquirer needs to specifically identify the required deliverables. A statement of objectives (SOO) is often used in performance-based contracting, where the acquirer indicates the results to be achieved, leaving the supplier to determine the processes, tools, and resources needed to deliver the service or product.

Detailed requirements statements are sometimes used for software procurements, but can be extremely complex for full specifications. Overly specifying may lead to much higher development costs when most of the essential features are already available in one or more commercially available software (COTS) products. However, without the critical features crisply identified, the likelihood of the software meeting the project needs is greatly reduced. Requirements may also include a list of the standards or specifications to which the system or service is required to conform. Lack of a detailed requirements statement at the onset of a project often leads to an adaptive strategy in which a few outcomes or scenarios are specified to get an acquisition project started.

A statement of work (SOW) details the work to be performed and is often appropriate for time-and-effort support with or without specific performance standards or service levels.

An SOW for software procurement typically includes the scope of contractual obligations for the contracted (i.e., bespoke) software and may include development processes to be used, metrics to be reported, and early delivery of product subset capabilities. (Section 12.1.3.2 of the PMBOK® Guide provides additional details concerning the Procurement SOW.)

Management requirements should be specified as well as technical requirements, including requirements for status reports, inputs for schedule updates, and participation in project meetings and reviews.

- 0**Establishing proposal evaluation criteria.** Criteria for evaluation procurement proposals submitted by potential suppliers (i.e., bidders) identify the supplier's specific technical capabilities, management approach, experience, and cost factors to be considered in selecting the supplier and explain how the factors will be weighted and evaluated. Usually, cost is not the most important factor in selecting a software or service provider, and within the cost factors, the initial cost is less important than the total cost over the project or product life cycle.

In addition to the evaluation criteria for the received proposals, the acquiring project manager needs to define the acceptance criteria or performance standard for the product or service to be provided. Acceptance of custom software may come, for example, on successful completion of acceptance testing by the users, or only after installation into the production environment.

- 0**Preparing terms and conditions.** Terms and conditions provide the details on how, where, and when the software product or service will be delivered. The supplier may have a preferred set of contract conditions related to cost, schedule, capabilities, maintenance, contract type, intellectual property rights, and data rights; the acquirer may also have a preferred set of contract conditions, which may be very different from those of the supplier. Therefore, it is critical that the software project manager participate in the development of the terms and conditions and understand their impact.

The upfront determination of an appropriate licensing approach (ownership of intellectual property and data rights) is critical in nearly all software procurements. Specifics as to who owns the product, non-compete clauses, warranties, and data management are a few of the issues to be resolved. It is important to objectively identify the project's needs for licensing rights and to determine the appropriate license type. The acquirer may develop a licensing strategy for computer software and require the vendor to provide a list of software products that have restrictions not explicitly stated in their commercial agreements. The licensing strategy should address four questions:

- Who needs to use the product or modify it and to what extent?
- What restrictions apply to accessing the supplied software by computer terminals and central processing units?
- What restrictions apply to transferring the supplied software to other parts of the buyer's organization and to sharing it with other parts of the organization?
- Are there any plans to incorporate the supplied software into the buyer's products?

Terms and conditions should include the type of contract, payment schedule, and expected period of performance. Since the acquired product or service needs to arrive in time for the overall project schedule, the software project manager should understand and account for the schedule risk associated with delivery of custom-built (bespoke) software by allowing a significant margin in the time for acquired software to be integrated with project-developed software.

- 0**Developing a procurement package.** A request for proposal (RFP) provides the necessary information to potential suppliers. It includes the technical requirements (SOO or SOW), the terms and conditions, the evaluation criteria, and instructions for bidders. The instructions explain what information should be included in the proposal, the anticipated procurement schedule, and the anticipated start date and period of

performance. Instructions often specify a maximum length for the proposal and the required structure of the proposal, so responses can be more easily compared. Software proposal instructions commonly require that proposals contain information needed to evaluate a bidder's capabilities and stability, such as descriptions of related projects, customer references, information on the qualifications of key personnel and staff certifications, and descriptions of facilities and technical resources.

12.1.1 Plan Software Project Procurement Management: Inputs

The inputs in Section 12.1.1 of the *PMBOK® Guide* are applicable for planning software procurement management.

12.1.1.1 Project Management Plan

See Section 12.1.1.1 of the *PMBOK® Guide*.

12.1.1.2 Requirements Documentation

See Section 12.1.1.2 of the *PMBOK® Guide*.

12.1.1.3 Risk Register

See Section 12.1.1.3 of the *PMBOK® Guide*.

12.1.1.4 Activity Resource Requirements

See Section 12.1.1.4 of the *PMBOK® Guide*.

12.1.1.5 Project Schedule

See Section 12.1.1.5 of the *PMBOK® Guide*.

12.1.1.6 Activity Cost Estimates

See Section 12.1.1.6 of the *PMBOK® Guide*.

12.1.1.7 Stakeholder Register

See Section 12.1.1.7 of the *PMBOK® Guide*.

12.1.1.8 Enterprise Environmental Factors

See Section 12.1.1.8 of the *PMBOK® Guide*.

12.1.1.9 Organizational Process Assets

See Section 12.1.1.9 of the *PMBOK® Guide*.

12.1.2 Plan Software Project Procurement Management: Tools and Techniques

The tools and techniques in Section 12.1.1 of the *PMBOK® Guide* are applicable for planning

software procurement management.

12.1.2.1 Make-or-Buy Analysis

See Section 12.1.2.1 of the *PMBOK® Guide*.

12.1.2.2 Expert Judgment

See Section 12.1.2.2 of the *PMBOK® Guide*.

12.1.2.3 Market Research

See Section 12.1.2.3 of the *PMBOK® Guide*.

12.1.2.4 Meetings

See Section 12.1.2.4 of the *PMBOK® Guide*.

12.1.3 Plan Software Project Procurement Management: Outputs

The outputs in Section 12.1.3 of the *PMBOK® Guide* are applicable for planning software procurement management.

12.1.3.1 Procurement Management Plan

See Section 12.1.3.1 of the *PMBOK® Guide*.

12.1.3.2 Procurement Statement of Work

See Section 12.1.3.2 of the *PMBOK® Guide*.

12.1.3.3 Procurement Documents

See Section 12.1.3.3 of the *PMBOK® Guide*.

12.1.3.4 Source Selection Criteria

See Section 12.1.3.4 of the *PMBOK® Guide*.

12.1.3.5 Make-or-Buy Decisions

See Section 12.1.3.5 of the *PMBOK® Guide*.

12.1.3.6 Change Requests

See Section 12.1.3.6 of the *PMBOK® Guide*.

12.1.3.7 Project Documents Updates

See Section 12.1.3.7 of the *PMBOK® Guide*.

12.2 Conduct Software Project Procurements

Section 12.2 of the *PMBOK® Guide* indicates that the primary activities in software project procurement include providing the procurement package to potential suppliers and communicating with them, receiving and evaluating offers, making a preliminary selection of one or more suppliers, and negotiating the agreement with the supplier.

For commercially available software packages, price can be the primary determinant, but the lowest proposed price may not be the lowest cost if the seller proves to be unable to deliver the products, services, or results in a timely and technically acceptable manner. Evaluations of suppliers should consider the supplier's project management practices and organizational stability. Ideally, the risk of supplier default on delivery will be low so that the burden on the buyer to manage and control the subcontractor will be minimized.

On reviewing the proposals, the acquirer may determine that the best choice for the project may be to modify the requirements in the SOO or SOW. Changes between the RFP requirements and the final agreement may be negotiated to support such considerations as affordability, timely completion of a basic set of software features, provision of named key personnel who are considered essential to performance of the work, reduced risk, alignment of the supplier's schedule with the project's master schedule, or additional tasks or functions and future upgrades. Negotiations may also address topics such as product acceptance, reporting, cost, usage, intellectual property rights, and data rights. One way of controlling risk is to add a contract provision for placing the source code into escrow, in the event of a contract dispute or dissolution of the supplier's organization.

12.2.1 Conduct Software Project Procurements: Inputs

The inputs in Section 12.2.1 of the *PMBOK® Guide* are applicable inputs for conducting software procurements.

12.2.1.1 Procurement Management Plan

See Section 12.2.1.1 of the *PMBOK® Guide*.

12.2.1.2 Procurement Documents

See Section 12.2.1.2 of the *PMBOK® Guide*.

12.2.1.3 Source Selection Criteria

See Section 12.2.1.3 of the *PMBOK® Guide*.

12.2.1.4 Seller Proposals

See Section 12.2.1.4 of the *PMBOK® Guide*.

12.2.1.5 Project Documents

See Section 12.2.1.5 of the *PMBOK® Guide*.

12.2.1.6 Make-or-Buy Decisions

See Section 12.2.1.6 of the *PMBOK® Guide*.

12.2.1.7 Procurement Statement of Work

See Section 12.2.1.7 of the *PMBOK® Guide*.

12.2.1.8 Organizational Process Assets

See Section 12.2.1.8 of the *PMBOK® Guide*.

12.2.2 Conduct Software Project Procurements: Tools and Techniques

The tools and techniques in Section 12.2.1 of the *PMBOK® Guide* are applicable for conducting software procurements.

12.2.2.1 Bidder Conferences

See Section 12.2.2.1 of the *PMBOK® Guide*.

12.2.2.2 Proposal Evaluation Techniques

See Section 12.2.2.2 of the *PMBOK® Guide*.

12.2.2.3 Independent Estimates

See Section 12.2.2.3 of the *PMBOK® Guide*.

12.2.2.4 Expert Judgment

See Section 12.2.2.4 of the *PMBOK® Guide*.

12.2.2.5 Advertising

See Section 12.2.2.5 of the *PMBOK® Guide*.

12.2.2.6 Analytical Techniques

See Section 12.2.2.6 of the *PMBOK® Guide*.

12.2.2.7 Procurement Negotiations

See Section 12.2.2.7 of the *PMBOK® Guide*.

12.2.3 Conduct Software Project Procurements: Outputs

The outputs in Section 12.2.1 of the *PMBOK® Guide* are applicable for conducting software procurements.

12.2.3.1 Selected Sellers

See Section 12.2.3.1 of the *PMBOK® Guide*.

12.2.3.2 Agreements

See Section 12.2.3.2 of the *PMBOK® Guide*.

12.2.3.3 Resource Calendars

See Section 12.2.3.3 of the *PMBOK® Guide*.

12.2.3.4 Change Requests

See Section 12.2.3.4 of the *PMBOK® Guide*.

12.2.3.5 Project Management Plan Updates

See Section 12.2.3.5 of the *PMBOK® Guide*.

12.2.3.6 Project Documents Updates

See Section 12.2.3.6 of the *PMBOK® Guide*.

12.3 Control Software Project Procurements

In addition to the issues addressed in Section 12.3 of the *PMBOK® Guide*, the following considerations apply to procurement of commercially available software (COTS) products. Because COTS products have frequent release cycles (typically 9 to 12 months), staying current requires an ongoing expenditure of resources to install and maintain the latest versions of COTS products. It is also helpful be aware of the likely evolution and life expectancy for a COTS product. The COTS provider may discontinue support of a COTS product and those who provided the expertise may be unavailable.

12.3.1 Control Software Project Procurements: Inputs

The inputs in Section 12.3.1 of the *PMBOK® Guide* are applicable for controlling software procurements.

12.3.1.1 Project Management Plan

See Section 12.3.1.1 of the *PMBOK® Guide*.

12.3.1.2 Procurement Documents

See Section 12.3.1.2 of the *PMBOK® Guide*.

12.3.1.3 Agreements

See Section 12.3.1.3 of the *PMBOK® Guide*.

12.3.1.4 Approved Change Requests

See Section 12.3.1.4 of the *PMBOK® Guide*.

12.3.1.5 Work Performance Reports

See Section 12.3.1.5 of the *PMBOK® Guide*.

12.3.1.6 Work Performance Data

See Section 12.3.1.6 of the *PMBOK® Guide*.

12.3.2 Control Software Procurements: Tools and Techniques

The tools and techniques in Section 12.3.2 of the *PMBOK® Guide* are applicable for controlling software procurements.

12.3.2.1 Contract Change Control System

See Section 12.3.2.1 of the *PMBOK® Guide*.

12.3.2.2 Procurement Performance Reviews

See Section 12.3.2.2 of the *PMBOK® Guide*.

12.3.2.3 Inspections and Audits

See Section 12.3.2.3 of the *PMBOK® Guide*.

12.3.2.4 Performance Reporting

See Section 12.3.2.4 of the *PMBOK® Guide*.

12.3.2.5 Payment Systems

See Section 12.3.2.5 of the *PMBOK® Guide*.

12.3.2.6 Claims Administration

See Section 12.3.2.6 of the *PMBOK® Guide*.

12.3.2.7 Records Management System

See Section 12.3.2.7 of the *PMBOK® Guide*.

12.3.3 Control Software Project Procurements: Outputs

The outputs in Section 12.3.3 of the *PMBOK® Guide* are applicable for controlling software procurements.

12.3.3.1 Work Performance Information

See Section 12.3.3.1 of the *PMBOK® Guide*.

12.3.3.2 Change Requests

See Section 12.3.3.2 of the *PMBOK® Guide*.

12.3.3.3 Project Management Plan Updates

See Section 12.3.3.3 of the *PMBOK® Guide*.

12.3.3.4 Project Documents Updates

See Section 12.3.3.4 of the *PMBOK® Guide*.

12.3.3.5 Organizational Process Assets Updates

See Section 12.3.3.5 of the *PMBOK® Guide*.

12.4 Close Software Project Procurements

While a software procurement activity usually ends before the software project ends, the need for the acquired service or software product may continue. Closing one procurement activity may signal the need to open another for ongoing maintenance. Another consideration is long-term technology relevance and the ability of the supplier to support technical change over time. If a software project manager plans to integrate a COTS product or custom-built software into their product, the project manager needs to be aware that the technology used to develop the COTS product or custom-built software could adversely affect the ability to make product enhancements in the future. Additional considerations for closing software procurements are provided in Section 12.4 of the *PMBOK® Guide*.

12.4.1 Close Software Project Procurements: Inputs

The inputs in Section 12.4.1 of the *PMBOK® Guide* are applicable for closing software procurements.

12.4.1.1 Project Management Plan

See Section 12.4.1.1 of the PMBOK® Guide

12.4.1.2 Procurement Documents

See Section 12.4.1.2 of the PMBOK® Guide

12.4.2 Close Software Project Procurements: Tools and Techniques

The tools and techniques in Section 12.4.2 of the *PMBOK® Guide* are applicable for controlling software procurements.

12.4.2.1 Procurement Audits

See Section 12.4.2.1 of the PMBOK® Guide.

12.4.2.2 Procurement Negotiations

*See Section 12.4.2.*2 of the *PMBOK® Guide.*

12.4.2.3 Records Management System

*See Section 12.4.2.*3 of the *PMBOK® Guide.*

12.4.3 Close Software Project Procurements: Outputs

The outputs in Section 12.4.3 of the *PMBOK® Guide* are applicable for closing software procurements.

12.4.3.1 Closed Procurements

See Section 12.4.3.1 of the *PMBOK® Guide*.

12.4.3.2 Organizational Process Assets Updates

See Section 12.4.3.2 of the *PMBOK® Guide*.

13 SOFTWARE PROJECT STAKEHOLDER MANAGEMENT

The introduction to Section 13 of the *PMBOK® Guide* states: Project Stakeholder Management includes the processes required to identify all people or organizations impacted by the project, analyzing stakeholder expectations and impact on the project, and developing appropriate management strategies for effectively engaging stakeholders in project decisions and execution.

The *PMBOK® Guide* also states that stakeholder management focuses on continuous dialogue with stakeholders to meet their needs and expectations, addressing issues as they occur, and fostering appropriate stakeholder engagement in project decisions and activities. Stakeholder satisfaction should be managed as a key project deliverable.

Most of the material in Section 13 of the *PMBOK® Guide* is applicable to software projects. This section of the *Software Extension to the PMBOK® Guide – Fifth Edition* presents additional considerations for managing stakeholders of software projects.

Figure 13-1 provides an overview of Software Project Stakeholder Management; it is an adaptation of Figure 13-1 in the *PMBOK® Guide*.

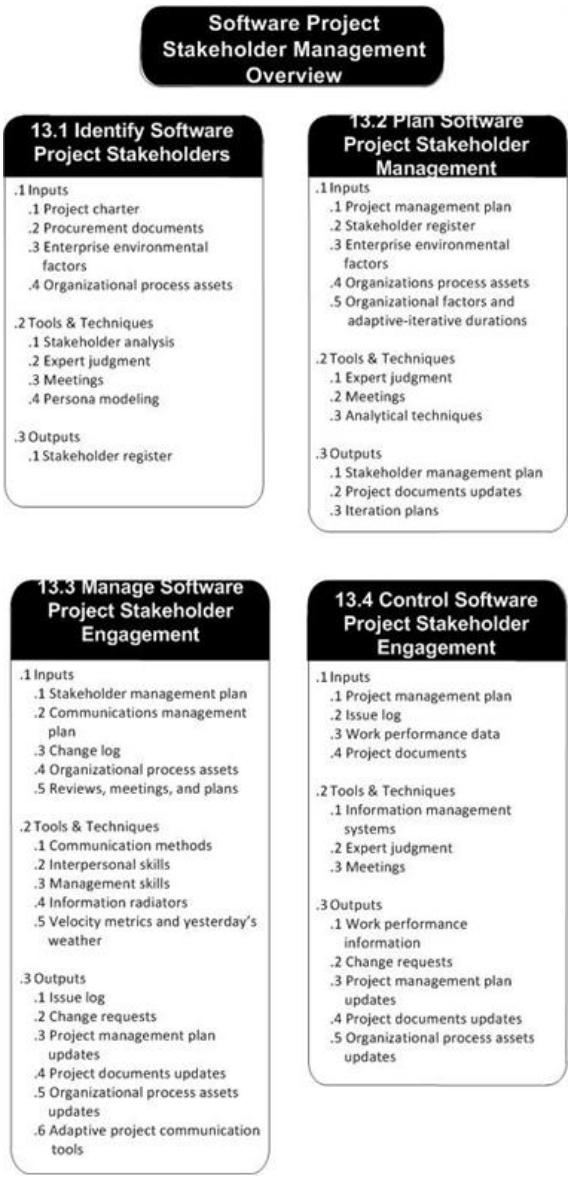


Figure 13-1. Software Project Stakeholder Management Overview

Stakeholder management is critical for achieving successful outcomes for software projects because software has no physical presence and is often novel. Software is difficult to visualize until it is demonstrated. In addition, there often exists a gulf of expectation between what a customer or product owner states and what the developer interprets. Misalignments among stakeholders represent a major risk for successful completion of

software projects.

Adaptive life cycles for software projects raise the visibility of the evolving product and the engagement of the customer and other stakeholders through frequent demonstrations, discussions, negotiations, and collaborative planning. As illustrated in Figure 13-2, predictive life cycle software projects have high stakeholder involvement at the beginning of the project when plans and requirements are being developed and at the end when user testing and product acceptance occur. Adaptive lifecycle software projects include frequent demonstrations of evolving increments of functional, deliverable software for the customer and other stakeholders, thus maintaining product visibility and stakeholder engagement throughout the duration of the software project.

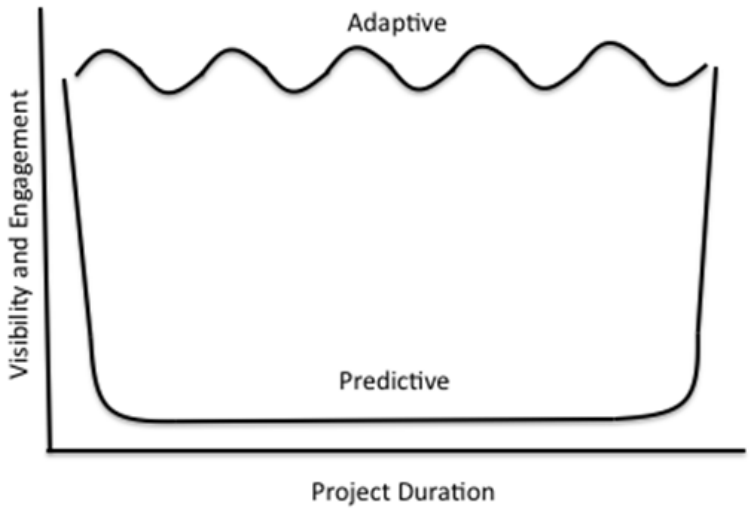


Figure 13-2. Product Visibility and Stakeholder Engagement

13.1 Identify Software Project Stakeholders

The material in Section 13.1 of the *PMBOK® Guide* is applicable to identifying stakeholders for software projects.

13.1.1 Identify Software Project Stakeholders: Inputs

The inputs in Section 13.1.1 of the *PMBOK® Guide* are applicable for identifying software project stakeholders.

13.1.1.1 Project Charter

See Section 13.1.1.1 of the *PMBOK® Guide*.

13.1.1.2 Procurement Documents

See Section 13.1.1.2 of the *PMBOK® Guide*.

13.1.1.3 Enterprise Environmental Factors

See Section 13.1.1.3 of the *PMBOK® Guide*.

13.1.1.4 Organizational Process Assets

See Section 13.1.1.4 of the *PMBOK® Guide*.

13.1.2 Identify Software Project Stakeholders: Tools and Techniques

The tools and techniques in Section 13.1.2 of the *PMBOK® Guide*, are applicable for identifying software project stakeholders. In addition, Section 13.1.2.4 is applicable for identifying software project stakeholders.

13.1.2.1 Stakeholder Analysis

See Section 13.1.2.1 of the *PMBOK® Guide*.

13.1.2.2 Expert Judgment

See Section 13.1.2.2 of the *PMBOK® Guide*.

13.1.2.3 Meetings

See Section 13.1.2.3 of the *PMBOK® Guide*.

13.1.2.4 Persona Modeling for Software Projects

Software teams sometimes use persona modeling to identify and analyze project stakeholders. Personas are visual summaries of key stakeholders and their interests. Personas may be real people or composites. They have the following characteristics: an archetypal description, grounded in reality; goal-oriented; specific and relevant; and tangible and actionable. They are not a replacement for requirements but instead augment and support prioritization of the requirements. Personas provide insight by providing focus, understanding, and empathy for the users of a system. An example of persona modeling is illustrated in Figure 13-3.

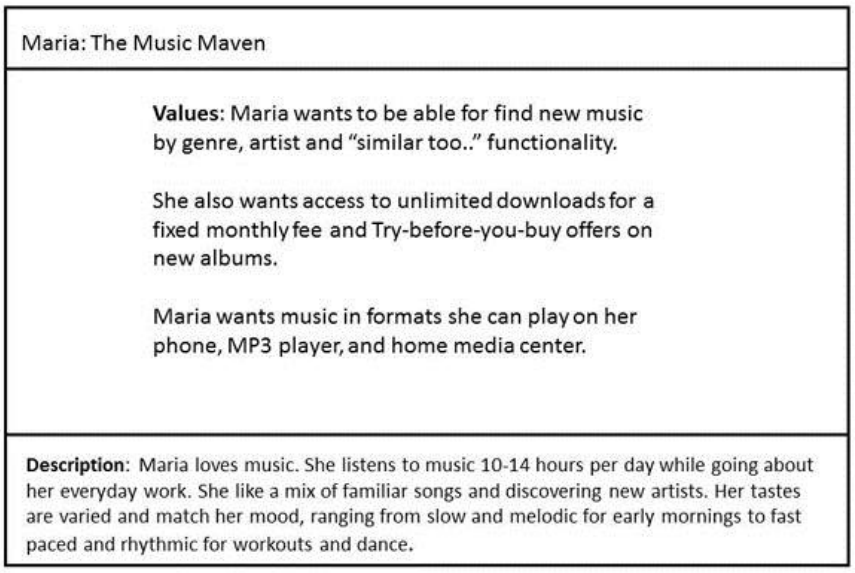


Figure 13-3. Depiction of Software Project Persona Modeling

Persona attributes may include goals, influencers, questions, and frustrations and pain points in addition to knowledge, activities, and interest. These attributes can be

tailored to software projects to provide descriptions of stakeholder personas and provide a basis for stakeholder analysis.

Personas support better decision making by keeping a team focused on delivery of value-adding product features. Teams can shorten discussions by referencing personas that team members are familiar with to settle issues of needs, wants, and exclusions.

13.1.3 Identify Software Project Stakeholders: Outputs

The output in Section 13.1.3 of the *PMBOK® Guide*, are applicable for identifying software project stakeholders.

13.1.3.1 Stakeholder Register

See Section 13.1.3.1 of the *PMBOK® Guide*.

13.2 Plan Software Project Stakeholder Management

As stated in the introduction to Section 13 of the *PMBOK® Guide,* stakeholder management also focuses on continuous dialogue with stakeholders to meet their needs and expectations, addressing issues as they occur, and fostering appropriate stakeholder engagement in project decisions and activities.

For software projects, it is especially important to plan for close and frequent involvement of customers and product owners (i.e., business-stakeholder involvement) to provide validation that the project is progressing towards the desired goal and that the evolving product is suitable, because software functionality and behavior are difficult to assess until demonstrated. The acronym IKIWISI (I'll Know It When I See It) is often used to describe this issue and highlights the need for frequent demonstrations and trials to fine-tune required (and desired) functionality and behavior. Because of the opportunity for divergent expectations among stakeholders and project team members, software project managers should plan for demonstrations in timeframes of weeks rather than months.

13.2.1 Plan Software Project Stakeholder Management: Inputs

The inputs in Section 13.2.1 of the *PMBOK® Guide*, are applicable for identifying software project stakeholders, with the modification of Section 13.2.1.3. An additional input for identifying software project stakeholders is provided in 13.2.1.5.

13.2.1.1 Project Management Plan

See Section 13.2.1.1 of the *PMBOK® Guide*.

13.2.1.2 Stakeholder Register

See Section 13.2.1.2 of the *PMBOK® Guide*.

13.2.1.3 Enterprise Environmental Factors for Software Projects

Bureaucracy, office politics, and personal dynamics lead to inefficiencies in decision-making and are an unavoidable input to stakeholder management. Although these factors are present in all business situations, these issues are typically exposed early in software projects that use adaptive project life cycles because team members interact closely with project stakeholders. These issues may be less visible when using a

predictive software project life cycle to manage a software project.

13.2.1.4 Organizational Process Assets

See Section 13.2.1.4 of the *PMBOK® Guide*.

13.2.1.5 Organizational Factors and Adaptive-Iteration Durations

Software projects that use adaptive life cycles and that have limited availability of stakeholders to review product increments, or that share resources with slow cadence projects will likely adopt iteration durations in the 2-4 week range. Projects without these restrictions will likely adopt iteration durations in the 1-2 week range. The software processes and tools used to release software to a test environment may also influence the duration of iterations.

13.2.2 Plan Software Project Stakeholder Management: Tools and Techniques

The tools and techniques in Section 13.2.2 of the *PMBOK® Guide*, are applicable for identifying software project stakeholders.

13.2.2.1 Expert Judgment

See Section 13.2.2.1 of the *PMBOK® Guide*.

13.2.2.2 Meetings

See Section 13.2.2.2 of the *PMBOK® Guide*.

13.2.2.3 Analytical Techniques

See Section 13.2.2.3 of the *PMBOK® Guide*.

13.2.3 Plan Software Project Stakeholder Management: Outputs

The outputs in Section 13.2.3 of the *PMBOK® Guide*, are applicable for planning software project stakeholder management. In addition, Section 13.2.3.3 is applicable to planning software project stakeholder management.

13.2.3.1 Stakeholder Management Plan

See Section 13.2.3.1 of the *PMBOK® Guide*.

13.2.3.2 Project Documents Updates

See Section 13.2.3.2 of the *PMBOK® Guide*.

13.2.3.3 Iteration Plans

For software projects that use adaptive life cycles, retrospectives that involve stakeholders in planning and reviewing activities that occur at the start and end of iterative cycles will be recorded in iteration plans.

13.3 Manage Software Project Stakeholder Engagement

Software projects that develop new and unprecedented software products are, or should be, collaborative explorations towards functionally and financially acceptable solutions. This rarely happens by accident and, in most cases, stakeholder engagement is actively managed to ensure that project objectives are stated and met. For adaptive software project life cycles, this takes the form of scheduled demonstrations and user trials of working, deliverable software at the end of selected iterations that produce increments of product capability.

When receiving feedback from customers, users, and other stakeholders following their evaluation of an increment of functionality, it may be tempting to interpret no comments as good news and not as issues or problems. Rarely is “no news” from stakeholders considered to be good news, especially early in a software project. Lack of feedback is more likely a sign of insufficient stakeholder involvement. Efforts should be made to ensure that project stakeholders seriously and thoroughly evaluate incremental versions of the product. If this does not occur, issues and desired changes may be discovered later in the project when the issues are more costly to fix, or it may be too late too incorporate requested changes while maintaining the delivery schedule.

13.3.1 Manage Software Project Stakeholder Engagement: Inputs

The inputs for managing project stakeholder engagement in Section 13.3.1 of the *PMBOK® Guide* are applicable for software projects. In addition, the input in Section 13.3.1.5 is applicable for software projects.

13.3.1.1 Stakeholder Management Plan

See Section 13.3.1.1 of the *PMBOK® Guide*.

13.3.1.2 Communications Management Plan

See Section 13.3.1.2 of the *PMBOK® Guide*.

13.3.1.3 Change Log

See Section 13.3.1.3 of the *PMBOK® Guide*.

13.3.1.4 Organizational Process Assets

See Section 13.3.1.4 of the *PMBOK® Guide*.

13.3.1.5 Reviews, Meetings, and Plans

Milestone reviews provide opportunities for stakeholder engagement in predictive life cycle software projects. Periodic technical interface meetings (TIMs) can also be held. For software projects that use adaptive life cycles, iteration plans provide an important input for managing software project stakeholder engagement. These plans provide an initial estimate of what will be included in each iterative release of the software. Retrospective meetings during each release demonstration provide the opportunity to dynamically update release plans.

13.3.2 Manage Software Project Stakeholder Engagement: Tools and Techniques

The tools and techniques in Section 13.3.2 of the *PMBOK® Guide* are applicable for software projects. In addition, Sections 13.3.2.4 and 13.3.2.5 are applicable to software projects.

13.3.2.1 Communication Methods

See Section 13.3.2.1 of the *PMBOK® Guide*.

13.3.2.2 Interpersonal Skills

See Section 13.3.2.2 of the *PMBOK® Guide*.

13.3.2.3 Management Skills

See Section 13.3.2.3 of the *PMBOK® Guide*.

13.3.2.4 Information Radiators

As explained in Section 10, information radiators are large, graphical displays of project status and the metrics used to report project status. They are frequently updated and located in view of the software project team and other project stakeholders. Commonly used graphs include task boards, burn-up and burn-down graphs, cumulative flow diagrams, and defect lists. Information radiators replace diffuse internal politics and unhealthy competition with project-relevant information. See Section 10 of this extension for examples of information radiators.

13.3.2.5 Velocity Metrics and Yesterday's Weather

See Section 10 of this software extension for a discussion of velocity metrics and yesterday's weather.

13.3.3 Manage Software Project Stakeholder Engagement: Outputs

The outputs for managing stakeholder engagement in Section 13.3.3 of the *PMBOK® Guide* are applicable for software projects. In addition, the output in 13.3.3.6 is applicable to managing stakeholder engagement for software projects.

13.3.3.1 Issue Log

See Section 13.3.3.1 of the PMBOK® Guide

13.3.3.2 Change Requests

See Section 13.3.3.2 of the PMBOK® Guide

13.3.3.3 Project Management Plan Updates

See Section 13.3.3.3 of the PMBOK® Guide

13.3.3.4 Project Documents Updates

See Section 13.3.3.4 of the PMBOK® Guide

13.3.3.5 Organizational Process Assets Updates

See Section 13.3.3.5 of the PMBOK® Guide

13.3.3.6 Adaptive Project Communication Tools

As discussed in Section 10 of this extension, adaptive life cycle models for software projects use a set of communications tools for describing scope, schedule, progress, and risks. These tools include product backlogs, release maps, cumulative flow diagrams, and product burn-down graphs, and risk burn-down graphs.

13.4 Control Software Project Stakeholder Engagement

Controlling and managing stakeholder engagement and expectations is arguably the single most important success factor for any software project manager. Techniques for managing adaptive life cycle software projects offer some unique challenges as well as some advantages over software stakeholder engagement techniques for predictive lifecycles. In particular, software project managers and the software team must have stakeholder acceptance of, and ongoing participation in the adaptive life cycle model being used.

Management and execution of adaptive life cycles should be explained to and supported by customers and other stakeholders. The software project team also needs to know what is expected of them when interacting with customers. Getting true buy-in by external stakeholders and by an inexperienced project team can be challenging and time consuming.

For adaptive life cycles, it is important that the customer and other decision-making stakeholders understand that they are responsible for feature identification, feature prioritization, and sequencing of development; that they control what gets worked on; and that they are to be provided with demonstrations on progress and product functionality, which will require their objective feedback. Controlling stakeholder expectations is much easier when this occurs than with major milestone reviews that are typical of stakeholder involvement in predictive life cycle projects. The development process becomes transparent and is easily adjusted as needed.

The inputs, tools and techniques, and outputs for controlling project stakeholder engagement in Section 13.4 of the *PMBOK® Guide* are applicable for controlling software project stakeholder engagement.

13.4.1 Control Software Project Stakeholder Engagement: Inputs

The inputs for controlling project stakeholder engagement in Section 13.4.1 of the *PMBOK® Guide* are applicable for software projects.

13.4.1.1 Project Management Plan

See Section 13.4.1.1 of the *PMBOK® Guide*.

13.4.1.2 Issue Log

See Section 13.4.1.2 of the *PMBOK® Guide*.

13.4.1.3 Work Performance Data

See Section 13.4.1.3 of the *PMBOK® Guide*.

13.4.1.4 Project Documents

See Section 13.4.1.4 of the *PMBOK® Guide*.

13.4.2 Control Software Project Stakeholder Engagement: Tools and Techniques

The tools and techniques for controlling stakeholder engagement in Section 13.4.2 of the *PMBOK® Guide* are applicable to controlling software project stakeholder engagement.

13.4.2.1 Information Management Systems

See Section 13.4.2.1 of the *PMBOK® Guide*.

13.4.2.2 Expert Judgment

See Section 13.4.2.2 of the *PMBOK® Guide*.

13.4.2.3 Meetings

See Section 13.4.2.3 of the *PMBOK® Guide*.

13.4.3 Control Software Project Stakeholder Engagement: Outputs

The outputs for controlling stakeholder engagement in Section 13.4.3 of the *PMBOK® Guide* are applicable for controlling software project stakeholder engagement.

13.4.3.1 Work Performance Information

See Section 13.4.3.1 of the *PMBOK® Guide*.

13.4.3.2 Change Requests

See Section 13.4.3.2 of the *PMBOK® Guide*.

13.4.3.3 Project Management Plan Updates

See Section 13.4.3.3 of the *PMBOK® Guide*.

13.4.3.4 Project Documents Updates

See Section 13.4.3.4 of the *PMBOK® Guide*.

13.4.3.5 Organizational Process Assets Updates

See Section 13.4.3.5 of the *PMBOK® Guide*.

Footnotes:

() The boldface numbers in parentheses refer to a list of references at the end of this standard.

(2) Available from http://www.sei.cmu.edu/
(3) Refactoring of software involves modifying the structure of a (partial or complete) software product without altering its behavior.
(4) The ConOps may also be known as a mission statement, a marketing analysis, or by some other name.
(5) ISO/IEC/IEEE Standard 1038 can be helpful in establishing peer review processes.
(6) Available from www.sei.cmu.edu/library/abstracts/reports/93tr006.cfm.

References

{ED NOTE: this is a rough draft of the references. The style and specific listings will be finalized prior to publication.}

[1] Project Management Institute, 2013. *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) Fifth Edition*. Newtown Square, PA: author.
[2] IEEE Standard 24765 (SEVOCAB)
[3] Project Management Institute, 2012. *PMI Lexicon of Project Management Terms.* Newtown Square, PA: author.
[4] PMI Code of Ethics and Professional Conduct
[5] Software Engineering Code of Ethics and Professional Practice;

http://www.computer.org/cms/Computer.org/Publications/code-of-ethics.pdf
[6] Association of Information Technology Professionals (AITP) Code of Ethics and Standards of Conduct;
http://c.ymcdn.com/sites/www.aitp.org/resource/resmgr/forms/code_of_ethics.pdf
[7] American Society for Information Science and Technology (AIS&T) Professional Guidelines; http://www.asis.org/AboutASIS/professional-guidelines.html
[8] ISO/IEEE Standard 15528 Systems and software engineering—System life cycle processes
[9] ISO/IEEE Standard 12207 Systems and software engineering—Software project life cycle processes
[10] SEI. November 2010. Capability Maturity Model Integrated for Development (CMMI-DEV V1.3); www.sei.cmu.edu/reports/10tr033.pdf
[11] SEI. November 2010. Capability Maturity Model Integrated for Services (CMMI-SVC V1.3); www.sei.cmu.edu/reports/10tr034.pdf
[12] Conway, M. "How Do Committees Invent?," *Datamation*, April, 1968.
[13] ISO/IEC/IEEE 16326: Systems and software engineering—Life cycle processes—Project management
[14] SEI. November 2010. CMMI for Acquisition (CMMI-ACQ V1.3); www.sei.cmu.edu/reports/10tr032.pdf
[15] IEEE Standard 16326 Systems and software engineering—Life cycle process—Project Management
[16] IEEE Standard 830-1998 Recommended Practice for Software Requirements Specifications
[17] IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements. Specifications
[18] IEEE Std 1362-1998 IEEE Guide for Information Technology—System Definition —Concept of Operations (ConOps) Document
[19] Fairley, R., *Managing and Leading Software Projects,* Wiley, Hoboken, NJ, 2009
[20] Tom DeMarco and Tim Lister: *Peopleware: Productive Projects and Teams*. Dorset House Publishing Company. New York, NY, 1999.
[21] Brooks, Frederick P. Jr., *The Mythical Man-Month, Anniversary Edition*, Addison Wesley, 1995
[22] Peter Drucker *Management Challenges for the 21st Century*, HarperBusiness; 1st edition (June 26, 2001)
[23] ISO/IEC 25010
[24] ISO Standard 9241-11: Guidance on Usability (1998)
[25] ISO/IEC/IEEE Standard 15026: Systems and software engineering – Systems and Software Assurance

[26] IEEE Std 1044 – Classification for Software Anomalies
[27] IEEE Standard 730 – Software Quality Assurance Plans
[28] IEEE Std 829 (Software and System Test Documentation);
[29] IEEE Std 1008 (Unit Testing)
[30] IEEE Std 1012 (System and Software Verification and Validation)
[31] IEEE Std 1028 – Software Reviews and Audits
[32] IEEE Standard 828 Configuration Management in Systems and Software Engineering
[33] Don Reinertson, *Managing The Design Factory*, Free Press (October 1, 1997)
[34] Paul Glen, David Maister, Warren Bennis. *Leading Geeks: How to Manage and Lead the People Who Deliver Technology*, Jossey-Bass (November 1, 2002)
[35] McGregor, D. *The Human Side of Enterprise*, McGrawHill. (1960).
[36] Tom DeMarco, Peter Hruschka, Tim Lister, Steve McMenamin, James Robertson, Suzanne Robertson *Adrenaline*
[37] *Junkies and Template Zombies: Understanding Patterns of Project Behavior* Dorset House (March 3, 2008)
[38] Bruce Tuckman, "Developmental Sequences in Small Groups" *Psychological Bulletin* 63 (1965): 384–99
[39] Patrick M. Lencioni, *The Five Dysfunctions of a Team*: A Leadership Fable (San Francisco: Jossey-Bass, 2002), 188–89.
[40] ISO Guide 73:2009, Risk Management: Vocabulary
[41] ISO/IEC 16085:2006, Standard for Software Engineering – Software project life cycle Processes – Risk Management