Chapter VI

Project Overall Planning

The will to succeed is important, but the will to prepare is even more important.
(Bobby Knight)

Getting off to a fast start in the right direction is important in any endeavor. This chapter discusses overall project planning and requirement analysis. These are two of the most important aspects of IT project management (Standish Group, 2004).

The Project Charter

In a previous chapter I discussed the project proposal and the project's business justification, which are often formalized in a project business plan. After these documents are approved, a formal project charter should be drafted. Whereas the project charter may be written by the project sponsor (project champion) or project manager (PM), the project charter is normally signed by general (upper line) management. The charter is the official go-ahead document for the project and indicates that funding and resources have been, or will shortly be, made available. The charter typically contains the following:

- Project title and description
- Project manager assigned and his or her authority level set (i.e., authority to set budget, schedule, staffing, procurement)
- Goals and objectives (what the project is to accomplish)

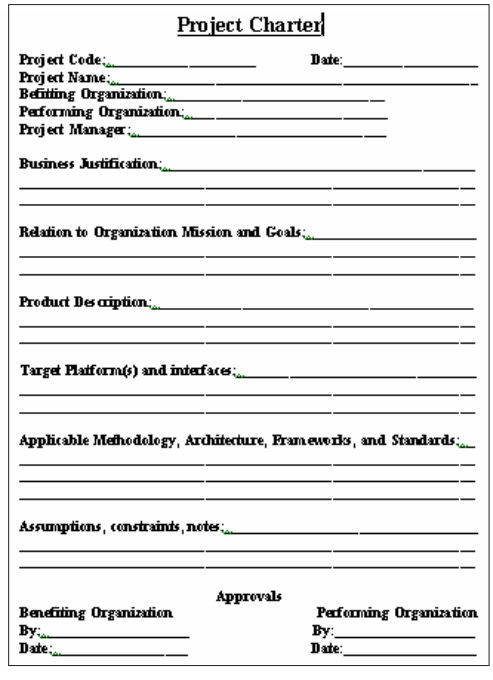
- Product (or service) description
- Applicable standards
- Assumptions and constraints

The charter should be signed by someone high enough in the organization so that everyone on the team will eventually report directly or indirectly to that person. Executives from both the performing and benefiting organization may sign off on the project charter, especially when both organizations are part of the same corporation. In larger corporations, IT project charters usually need the approval of the chief financial officer (CFO) and the chief information officer (CIO). The benefits of this charter are that it:

- Gives authority to the PM
- Formally recognizes the creation and existence of the project
- Outlines the objectives of the project

The charter should be broad enough that it will not require change during the project execution. Figure 6.1 shows an example of a simple project charter. If a project proposal or business plan is developed after the charter, the charter may also include financial information, such as the basic project budget and contingency funds.

Figure 6.1. Project charter



Project Charter

Project Code: _____ **Date:** _____
Project Name: _____
Benefiting Organization: _____
Performing Organization: _____
Project Manager: _____

Business Justification: _____

Relation to Organization Mission and Goals: _____

Product Description: _____

Target Platform(s) and interfaces: _____

Applicable Methodology, Architecture, Frameworks, and Standards: _____

Assumptions, constraints, notes: _____

Approvals

Benefiting Organization	**Performing Organization**
By: _____	**By:** _____
Date: _____	**Date:** _____

The Project Master Plan

Once the project charter has been signed off and a PM has been assigned to the project, an overall project master plan is assembled by the PM and his or her staff. The word *assembled* is used here because, in most organizations that have formal project management, much of the initial master plan is boilerplate material in which existing templates for the subplans are used and customized for the project at hand. This is often coordinated by a project management office (PMO). The PMO is discussed in detail in Chapter XVI.

The master plan may be a simple one-page document, as is shown in Figure 6.4, but it is typically a collection of subplan templates, as is illustrated in Figure 6.2. Each of these subplans incorporate (or simply refer to) policies, procedures, and standards for the organization as a whole. The formality and detail of this master plan should be based on the size and complexity of the project, as is illustrated in Figure 6.3. The content and nature of each of these subplans is described and illustrated in later chapters of this book.

Figure 6.2. Project master plan

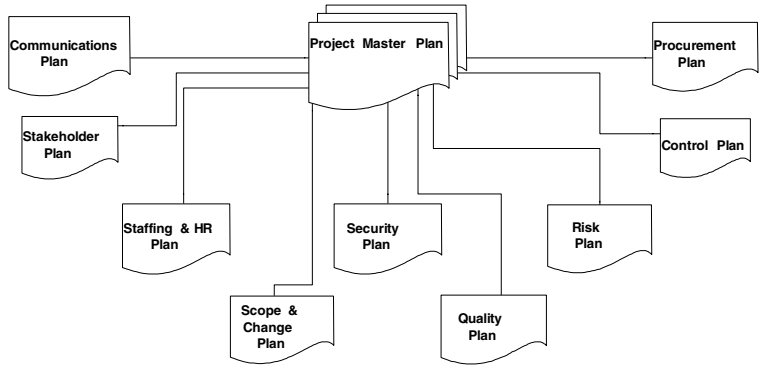


Figure 6.3. Master plan formality

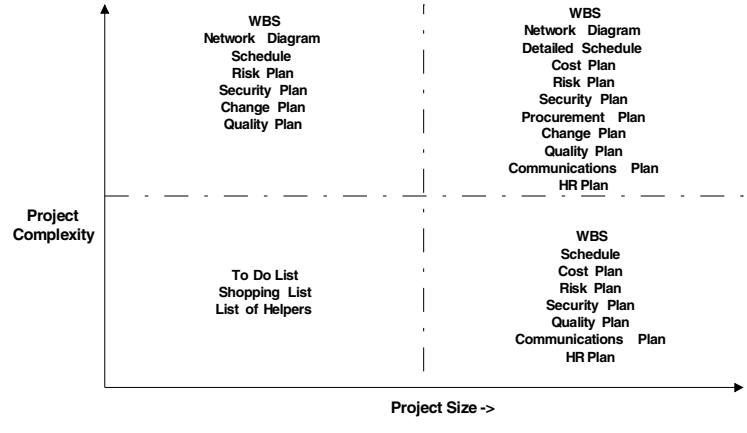


Figure 6.4. Project overall plan

Project Overall Plan

Project Code:_____ **Date:**_____
Project Name:_____
Befitting Organization:_____
Performing Organization:_____
Project Manager:_____

Rough Estimates: Start Date:_____ **End date:**_____ **Cost:**_____

Overall Product Deliverables – In Scope:

1. _____
2. _____
N. _____

Product Features – Out of Scope:_____
1._____
2._____
N._____

Key Milestones:

1. _____
2. _____
N. _____

Key Risks, Procurement, Security, and Quality Issues:_____
_____
_____

Key Stakeholders and Human Resources:

Person	Role/Responsibility	Contact Info
1.		
2.		
N.		

Communications and Reporting:_____
_____

Notes: _____
_____

Approvals

Benefiting Organization	Performing Organization
By:_____	By:_____
Date:_____	Date:_____

As the project unfolds and the scope is determined, the project-specific work and deliverables are incorporated into each of these subplans. This planning process, and the appropriate methods thereof, for IT projects are discussed and detailed in later chapters of this book.

The Software Engineering Institute's (SEI; www.sei.cmu.edu/cmm) CMM defines necessary Level 2 practices for software project planning:

- Are estimates documented for use in planning and tracking the project?
- Do the plans document the activities to be performed and the commitments made fore the project?

- Do all affected parties agree to their commitments?
- Does the project follow a written policy for planning a project?
- Are adequate resources provided for planning the project?
- Are measurements used to determine the status of the planning activities?
- Doe the PM review the activities for planning the project on both a periodic and event-driven basis?

The IEEE also has a standard for software project management plans: IEEE Std. 1058-1998. In their standard, the elements of such a management plan include:

Overview, references, and definitions

Project organization

- External interfaces
- Internal structure
- Roles and responsibilities

Managerial process plans

- Start-up (estimation, staffing plan, resource plan, budget plan)
- Work plan (activities, schedule, resources, budget)
- Control plan (requirements, schedule, budget, quality, reporting, metrics)
- Risk management plan
- Closeout plan

Technical process plans

- Process model
- Methods, tools, techniques
- Infrastructure plan
- Product acceptance plan

Supporting process plans

- Configuration management plan
- Verification and validation plan
- Documentation plan
- Quality assurance plan
- Reviews and audits
- Problem resolution plan
- Subcontractor management plan
- Process improvement plan

Additional plans

Modern IT projects should also have a specific security plan. Neither the current PMI nor IEEE standards include such a component in planning. The security issue in IT project management is twofold:

- Being able to shield the project work and project workers and other project resources from security threats
- Being able to build adequate security protection into the product that is the subject of the project

The first security item would also be included in the risk management plan, detailed in a later chapter of this book. A Gartner research report projected that IT project downtime due to security issues would rise from 5% in 2004 to 15% in 2008 for organizations that do not have a comprehensive security plan (Alexander, 2004). Such security plans should address both logical and physical security. However, it is no longer sufficient simply to "secure the perimeter" physically and logically; active security procedures need to be implemented for those objects already inside of the perimeter. Thus, security plans are also related to human resource (HR) planning in terms of procedures that may be necessary, such as personnel background checks; this is detailed later in the book. The second security item should be addressed (at least partially) in the software engineering that is embedded in the product, which is the subject of the project. *In today's IT environment, it is vital that both of these security points be fully addressed in the project planning.*

Project Calendars and Fiscal Periods

Before starting the project and the detail planning thereof, it is necessary to establish a project calendar. Such a calendar indicates the quantum of time used for both planning and reporting, as well as periods of nonwork, such as holidays and weekends. Some practitioners use scheduling systems whereby arbitrary time units can be used for task start and end dates. Also, some practitioners try to schedule IT resources down to the day or even hour. *For IT projects (and other types of professional work) this is inappropriate and ineffective; and as a result one may spend more time managing the schedule than managing the work.* IT human resources are largely professional types, they may work varying numbers of hours per day, they may be called upon to help another person or another project from time to time, and they may take off a day or two for whatever reason whenever they so chose. IT effort, time, and cost estimates involve considerable uncertainty, thus for all these reasons it is more effective to chose the project time quantum at a larger interval than 1 day;, periods of 1 week, 2 weeks, or a month are more appropriate.

In addition, the time quantum should match both the fiscal calendar of the organization and the accounting periods of the organization. This facilitates cost and completion reporting because such reporting can be incorporated into an existing payroll and/or

timekeeping process. The time period of 1 week is the commonly used quantum for professional projects. This is discussed in more detail in later chapters of this book.

Figure 6.5 shows an example of a fiscal calendar for the first half of 2004. In this calendar, week numbers are used as the basis for planning and reporting, and these week numbers are numbered sequentially for the year, as 2004-06, which is in fiscal month 2004-02. When a specific task is defined and assigned resources, the task is scheduled for one or more fiscal weeks. Figure 6.6 shows such a fiscal calendar for the NASA Jet Propulsion Laboratory.

Kickoff Meeting

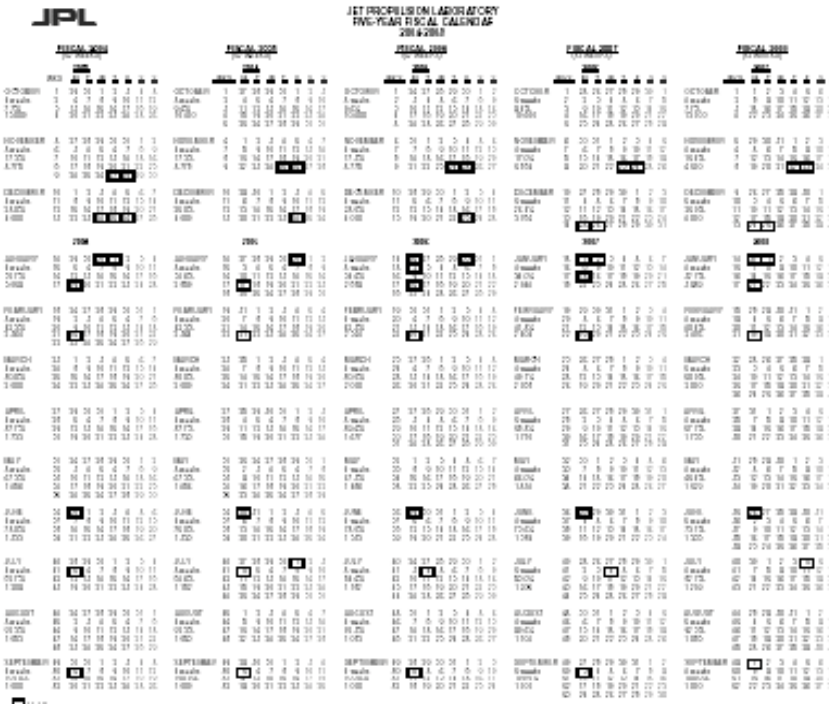
After the project charter has been approved and appropriately signed off, it is recommended to have an official kickoff meeting with the project team. This meeting has many benefits including finding out early if there are any major problems that have not surfaced already in the initial planning. Problems may include forgotten stakeholders or key team members, organizational issues, interpersonal issues, technical issues, environmental or regulation issues, or other constraints. Other alternative methods, plans, or approaches may also be discovered at this meeting. This first meeting allows team members to meet each other and builds enthusiasm, shared vision, and common goals and purpose. Things that should be done at this meeting include:

Figure 6.5. Project periods

Fiscal Year - 2004

Fiscal Month	Fiscal Week	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
01	01	5	6	7	8	9	10	11
	02	12	13	14	15	16	17	18
	03	19	20	21	22	23	24	25
	04	26	27	28	29	30	31	1
02	05	2	3	4	5	6	7	8
	06	9	10	11	12	13	14	15
	07	16	17	18	19	20	21	22
	08	23	24	25	26	27	28	29
03	09	1	2	3	4	5	6	7
	10	8	9	10	11	12	13	14
	11	15	16	17	18	19	20	21
	12	22	23	24	25	26	27	28
	13	29	30	31	1	2	3	4
04	14	5	6	7	8	9	10	11
	15	12	13	14	15	16	17	18
	16	19	20	21	22	23	24	25
	17	26	27	28	29	30	1	2
05	18	3	4	5	6	7	8	9
	19	10	11	12	13	14	15	16
	20	17	18	19	20	21	22	23
	21	24	25	26	27	28	29	30
06	22	31	1	2	3	4	5	6
	23	7	8	9	10	11	12	13
	24	14	15	16	17	18	19	20
	25	21	22	23	24	25	26	27
	26	28	29	30	1	2	3	4

Figure 6.6. NASA JPL calendar



- Introduce the project sponsor or champion
- Establish clear leadership on the project
- Share then vision of the champion and other leaders
- Clarify and communicate goals and objectives to team and stakeholders
- Discuss overall project plan
- Review major milestones and deliverables
- Establish working relationships and lines of communications
- Explain relevant policies and procedures
- Get teams members to know one another
- Review status to date
- Review standards that will apply
- Establish responsibilities (individual and group)
- Make sure everyone understands his or her role and tasks
- Solicit questions and comments
- Document and follow up on questions that cannot be answered at this time
- Identify potential problems and risks
- Handle any other issues that may interfere with starting work
- Give team formal go-ahead to start work

For this meeting, one should follow the general guidelines for effective meetings; these guidelines are discussed in Chapter XIII.

Scope Management

A project's scope is the work to be done and the things to work on. This scope is enclosed within a multidimensional boundary line that separates those things that are part of the project from other things that are not part of the project. As part of defining this boundary line, one may itemized the things that are to be included in the project work and deliverables as well as to itemize those things that are not part of the project work. This itemization may be at a general level or a very specific level, or a combination thereof. The dimensions may include what, how, who, when, and other concepts used as metrics or definitions.

According to PMI's PMBOK (PMI, 2000), scope management involves the following processes:

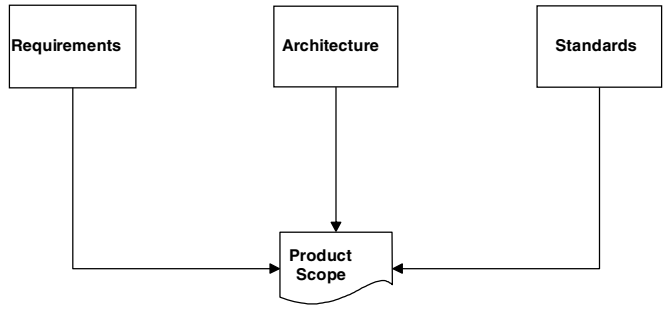
- Scope Initiation
- Scope Planning
- Scope Definition
- Scope Verification
- Scope Change Control

Scope initiation, as discussed earlier, involves making sure that the project charter is approved and that the necessary financial and other resources are available to move forward with the project. Scope planning involves developing a written scope statement that is more specific than that included in the project charter. The scope statement usually further itemizes and defines the project objectives and includes the major features of the IT product(s) and other major deliverables.

For IT projects, the project scope is more extensive than the product scope. The *product scope* will be eventually be described by design documentation and includes the features and specific functionality of the software product(s) that is to be built, procured and/or integrated as well as appropriate standards and the chosen architecture. This is illustrated in Figure 6.7.

The *project scope* includes not only the product(s) but also all of the associated activities and deliverables. Some deliverables may be called for in the requirements, some deliverables are mandated by the organization's chosen methodology and/or standards, and some deliverables are called for in the contract between the performing organization and benefiting organization. The aforementioned project charter describes the project scope at a very high level, and often the charter may only describe the product scope.

Figure 6.7. Product scope



A separate deliverable definition document (DDD) may be produced, which lists every deliverable, the form of the deliverable (document, software, hardware, etc.), applicable standards, and approval levels. Items typically found in the DDD include:

- Requirements document
- Overall design document
- Paper prototype (storyboards)
- Detail design document
- Product prototype
- Users' manual (external specifications document)
- Internal specifications document
- Test plan and scripts
- The product itself
- Installation and operation document

The scope definition process involves determining the all the details embodied within the scope statement and DDD and then subdividing those details into smaller, more manageable components. For IT systems, this first major activity of determining the scope details is called requirements analysis, which is discussed later in this chapter. Those detail requirements must be subdivided further in order to effectively and accurately plan and control the project. The further definition (or breakdown) of scope is manifested in a work breakdown structure (WBS), which is defined by PMI as "a *deliverable oriented* grouping of project components which *organizes* and defines the total scope of the project" (PMI, 2000).

The WBS is typically arranged in a multilevel hierarchical manner, and the first level (Level zero) often corresponds to the project life cycle (phases: requirements, design, construction, etc.) and is fully specified before the project is further broken down; each level of the WBS is a further breakdown of the higher level. The lowest level defines the

individual tasks, work packets, or activities. The development of WBS's for IT projects is discussed in detail in Chapter VII.

Scope verification is the process of formalizing the acceptance of the project by the stakeholders, particularly the benefiting organization (customer). It involves review of the work results (the final product or service) versus the scope definition (requirements). Scope verification

is typically done at the end of each project phase or, as suggested in this book, at quality stage gates; it involves requirements traceability, verification, and validation. Verification is a process related notion and is one of our critical completion success factors. It answers the general question, Have we done this process correctly? or, more specifically, Have we built the product correctly? Validation is a product-related notion and is one of our satisfaction critical success factors. It answers the general question, Have we done the correct process? or, more specifically, Have we built the correct product (the product the customer wanted)? Traceability of requirements means that every requirement is properly included in every preliminary product manifestation as well as in the final product; traceability is actually a part of validation. This is discussed in detail in Chapter X.

Scope change control is concerned with influencing the factors that cause change and controlling them to ensure that changes are beneficial. Change control also happens in retrospect to determining that a scope change has actually occurred and then handling the change in an appropriate manner. As part of this scope planning process, a scope management plan may be formally written that indicates how the scope will be further defined and then later managed in regard to scope changes. A scope change control process and system may be implemented that defines the procedures by which the project scope may be changed and often includes:

- Forms and other paperwork
- Tracking systems
- Approval levels
- Billing and or contract change procedures

The scope management plan also includes the processes necessary to make sure that all project work is addressed and extra work is not done. This extra work, which is outside of the project scope, is called gold plating and should not be done without the proper written approval of the benefiting and performing organizations. The goal of project management is to give the customer what they have asked for and expect, no more and no less. Most IT projects do not fully succeed, and so extra work should not be done; after all, one does not know that the customer actually desires that extra work or not, or if the customer will pay for that extra work. In this book, we address scope changes under the larger topic of "change management," which is discussed in detail in Chapter XI.

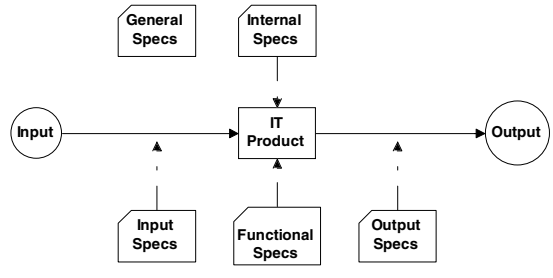
Requirements Analysis

One of the best and quickest ways to ensure problems or failure in an IT project is to have hastily gathered, ill-defined, incomplete, and/or poorly documented requirement specifications. Requirements are the details describing an IT product's features and other properties that are needed (and wanted) by the benefiting organization. Requirements should embody the features and capabilities of the IT product (application) and also the behavior and content of the application. The requirements analysis process involves several subprocesses, including:

- Requirement discovery
- Requirement organization and documentation
- Requirement prioritization and project phasing
- Requirement change management

Requirements can take many forms, and this is illustrated in Figure 6.8. One form is input specifications, which describe the content and form of the input to the application. Another form is output specifications, which describe the content and form of the output. For interactive systems, these input and output specifications are often combined into external specifications or interface requirements. The processing that the application will do is described in a set of functional specifications, and how that processing is done is specified in a set of technical or internal specifications. The external specifications will be externally observable in the application, and the functional specifications may be externally observable or implied from the output versus the input. However, the internal specifications may not be externally observable. Even though not externally observable, these internal specifications may be vital to the long-term success of the application, including its maintainability and adaptability. As discussed in Chapter V, many of these internal requirements can be embodied in the set of software standards adopted by an organization.

Figure 6.8. Types of requirements



General specifications can involve any other properties of the application or the application's environment. As is illustrated in Figure 6.7, standards and target architecture may be separated from the application requirements, as the standards and architecture may be common to a number of applications, perhaps all the applications that are built/integrated by a particular organization. There is an IEEE standard for requirements (830), and that standard differentiates between product requirements and project requirements. Project requirements typically involve date/time and cost issues, and these are usually specified separately in contracts from the requirements or the statement of work (SOW); contract issues are discussed in Chapter XII.

Other general specifications may include grade and quality levels, portability, load and scalability, response times and other service levels, information content, security, and constraints such as the use of certain products or the use of general products as commercial off the shelf (COTS). Becoming more important in modern times is the need for security requirements to be detailed as early as possible. Security cannot be an afterthought, because good security has to be built into the total product (and the methodology of constructing it), not added on later. New legislation in various governments may be mandating added security-related requirements such as the Health Insurance Portability Act (HIPPA) and Sarbanes-Oxley Act (SOX) laws in the United States.

Requirement discovery is a process that involves a cooperative effort between the performing organization and the benefiting organization. This process may also involve other stakeholders with different needs and priorities. Chapter XIII deals with stakeholder identification and analysis, and how to handle different types of stakeholders. The IT personnel involved with requirement discovery are usually analysts such as business analysts, systems analysts, market analysts, or so-called requirements analysts. These analysts facilitate the flow of information from the users of the application to the builders of the application (designers and programmers). Thus these analysts need a combination of both people skills and technical skills; human resource and communication issues are also detailed in Chapter XIII.

Requirements must be consistent with the stated purpose of the overall system, and the PM must referee the requirements discovery process to make sure that suggested requirements are consistent with the project charter. Requirements should also be clear, complete, sufficiently detailed, doable, and testable. Just as it may be of little use to have a great solution to the wrong problem, it is of little practical use to build the wrong product very well. For this reason, requirements have to be clear (unambiguous), complete, and detailed enough so that the product, or at least a preliminary product manifestation, can be constructed. For example, a common nontestable requirement that often shows up in requirement specification is that the application should be user friendly. This is too subjective, and a testable version of this notion should be included instead, perhaps involving the types of human-computer interaction and the devices involved (mouse, keyboard, etc.).

There are many techniques of requirement discovery for IT projects, and these include:

- Informal conversations with end users and/or their management
- Structured interviews with end users and/or their management

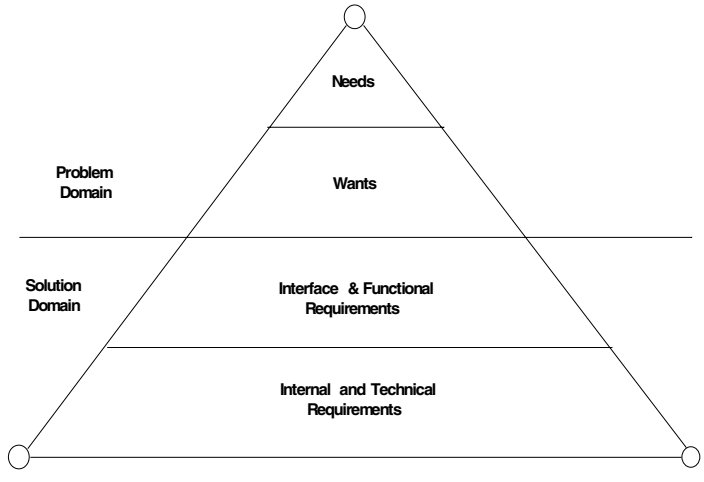
- Collaborative working sessions between the benefiting and performing organizations
- Review of existing work flow and work processes
- Review of existing IT systems
- Review of existing external or internal documentation of processes and/or systems

Care should be taken to involve all of the stakeholders at some point in the total requirements process. Anyone who has an interest in the project and is not involved to some degree in the requirement discovery process could later cause problems. Again, stakeholder determination, analysis, and planning are discussed in detail in Chapter XIII.

A benefiting organization's requirements usually evolve during the planning and execution phases of a project. Requirements usually evolve from descriptions of the problem to descriptions of the solution; this is illustrated in Figure 6.9.

"The phenomenon is likened to a continuous application of Maslow's hierarchy of needs. Every time any need is satisfied, more needs appear" (Davis, Hickey, & Zweig, 2004). Often, end users cannot express in words what they want, but they may know what they want once they see it, or, conversely, they may know what they *do not* what when they see it. Showing the end users preliminary product manifestations is the best way to flush out all the real requirements, to further define the requirements, and to expedite this overall requirement evolution process. These preliminary product manifestations are part of the overall validation process, and those manifestations used depend upon the choice of methodology, as discussed in Chapter XVI. Typical manifestations include paper prototypes, working prototypes, and user's manuals. Users' review of these preliminary product manifestations is part of the quality stage gate process, which is described previously in this book and will be elaborated upon in upcoming chapters.

Figure 6.9. Requirements evolution



Organizing requirements in an effective and efficient manner can be difficult. One wants to cover as much detail as possible, but the resulting document must be clear. In addition, the time to produce the requirement document should not consume a high percentage of the project time and budget, and the document must not be overly burdensome to work with. Different organizations vary considerably in the manner and form of producing this document. Forms of the requirements document may be:

- No document at all (only in the minds of the users and developers)
- Simple checklist
- Spreadsheet
- Database
- Formal document
- High-level design documents (i.e., UML use cases)
- Specialized software product

Combinations of documents in this list may also be used. The degree of formality of this documentation process depends upon the software methodology chosen and the type of formal or implied contract between the performing and benefiting organizations. Whether the contracting situation between the performing and benefiting organization is formal or not, the requirements should be agreed to in writing by the line management of both organizations as well as the PM. Usually in the business world, "if it is not written, it never existed and/or was never agreed to."

As part of the validation process, requirements must be traceable through all preliminary product manifestations and in the final product and test plan thereof. Figure 6.10 is an example of a spreadsheet form of a requirements document that provides for each requirement: a brief description, a unique identifying code, a reference to a more detail description, and checks that each appears in the product manifestations. The spreadsheet may be sorted based on the contents of any column(s) such as priority, type, or appearance in the product manifestations.

Often, requirements are represented with high-level design documents. UML "use cases" are used more because their visual format facilitates communication with end users. A use case clearly shows the external aspects of a system. In the use case diagram, "actors," which may be humans or other IT systems, are shown as stick figures. The actors make requests of the system, provide information to the system, and receive information from the system. A rectangle, which represents the system's boundary, is drawn, and processes are shown as ovals within the rectangle. The processes respond to the actor's requests by either providing information back to the actor and/or changing the state of the system (information content of the system). An example is shown in Figure 6.11 with multiple processes. In addition to the drawing, there would be a simple text description for each interaction (line) between the actors (stick figures) and a process (oval) within the system (rectangle). These text descriptions embody both the interface specifications (what and how information is exchanged between the system process and the actor) and functional specifications (what the process is supposed to do); how the

process works is not included in the use case. An excellent method of documenting external requirements is to combine the spreadsheet of Figure 6.10 with use cases, wherein the last column in each row of the spreadsheet references an interaction in a use case diagram.

Use cases (or a similar technique) are also a good discovery method because they keep unnecessary detail out of the requirement process. As previously stated, *how* the processes within the system work should usually not be part of the discovery process. The how of the process is determined later in the design stage (or detail design stage) of the project, including the definition of data requirements (database tables), classes, packages, algorithms, and so forth. A general misconception is that use cases are only used in object-oriented systems. Although I feel that object-oriented methods should always be used for new IT systems, use cases can be used just as effectively for non–object-oriented systems. Paper prototypes or "storyboards" are often used as a follow up to use cases or other forms of preliminary requirement documentation to further discover or clarify/detail requirements. These have many of the same advantages of live prototypes, but they can be produced and modified quicker and cheaper. Paper prototypes can be constructed with paper and pencil, simple drawing programs, presentation software, or RAD products. Storyboards essentially walk the user through the interaction between the end users and the system and visually show that interaction through screen mock-ups of both input and output screens. Each interaction in a use case diagram could be simulated via a storyboard.

Note that some software development methodologies may use little or no requirements documentation. Agile methods, discussed earlier in this book, use methods that require very close interaction between developers and end users to develop incrementally the application and to produce no formal and very little informal requirement documentation. Often the programmers use a test first approach to create automated testing scripts that embody the requirements in those test scripts. As discussed in Chapter V, these agile methods may have limited application.

After the initial requirements have been discovered and recorded, it must be determined which of those requirements will actually be included in the project or in the first phase of the project. First, the requirements must be prioritized, then very rough estimates of the time/cost of each requirement must be developed, typically using historical (or industry available) costs of similar work. In the requirements spreadsheet shown in Figure 6.10, another column could be added for the estimated cost and/or time. Detail cost estimation techniques are covered in Chapter VII. Both the benefiting organization and performing organization are integral to this process, which involves trade-offs and compromises between the project dimensions of time, cost, and scope. This requirement selection process may be done in a number of ways, and the particular manner in which it is accomplished is related to the chosen software engineering methodology and contracting arrangement to be used as

- Single-phase, fixed scope
- Multiphase, fixed scope per phase
- Multiphase, fixed time per phase
- Multiphase, fixed cost (budget) per phase

The first two methods are best suited to contract and bid situations, and the last two methods may be best suited to a type of incremental or iterative development methodology. Other issues may also factor into requirement phasing such as risks, market and application timing, interdependent applications, and available resources. In the requirements spreadsheet shown in Figure 6.10, another column could be added for the project phase; or separate spreadsheets (perhaps in the same workbook) could be used for each phase.

The final requirement process is "requirement change management." Requirements will evolve—some requirements will change, some will be removed, and others will be added. At some point in the initial planning process, the baseline requirements must be set (at least for the first phase), and after that, formal change control should be used. Formal change control is necessary for proper coordination between all the stakeholders, to make sure that all changes are needed and wanted, to review the total impact of changes, and to make sure that changes are properly included in the financial arrangements between the benefiting organization and performing organization. Changes to requirements should go through the same process as the original requirements, including written approval, cost/time impacts, project schedule revision, and traceability. Change control is discussed in detail in a later chapter of this book. The PM must make sure that all stakeholders know the full impact that changes may have on the project, and manage both the changes and stakeholders in the manner most beneficial to the project.

The Software Engineering Institutes Capability Maturity Model includes requirements management as a key process in its Level 2 maturity (www.sei.cmu.edu/cmm).

Level 2 Key Process Areas:

- Requirements Management
- Software Project Planning
- Software Project Tracking and Oversight
- Software Subcontract Management
- Software Quality Assurance
- Software Configuration Management

Figure 6.10. Requirements specification form

Requirements Specification

			Type				Traceability						
REQ ID	Description	Priority	External	Functional	Internal	General	Overall Design	Detail Design	Paper Prototype	Working Prototype	Test Plan	Users Manual	Detail Reference

The level 2 requirements management goals are:

- System requirements allocated to software are controlled to establish a baseline for software engineering and management use.
- Software plans, products, and activities are kept consistent with the system requirements allocated to software.

The key practices under each common feature for requirements are:

Commitment to Perform

The project follows a written organizational policy for managing the system requirements allocated to the software.

Ability to Perform

For each project, responsibility is established for analyzing the system requirements and allocating them hardware, software, and other system components

The allocated requirements are documented

Adequate resources and funding are provided for managing allocated requirements

Members of the software engineering group and other software-related groups are trained to perform their requirements management activities

Activities Performed

The software engineering group reviews the allocated requirements before they are incorporated into the software project

The software engineering group uses the allocated requirements as the basis for the software plans, work products, and activities

Changes to the requirements are reviewed and incorporated into the project

Measurement and Analysis

Measurements are made and used to determine the status of the activities for managing the allocated requirements

Verifying Implementation

The activities for managing the allocated requirements are reviewed with senior management on a periodic basis

The activities for managing the allocated requirements are reviewed with the project manager on both a periodic and event-driven basis

The software quality assurance group reviews and/or audits the activities and work products for managing the allocated requirements and reports the results

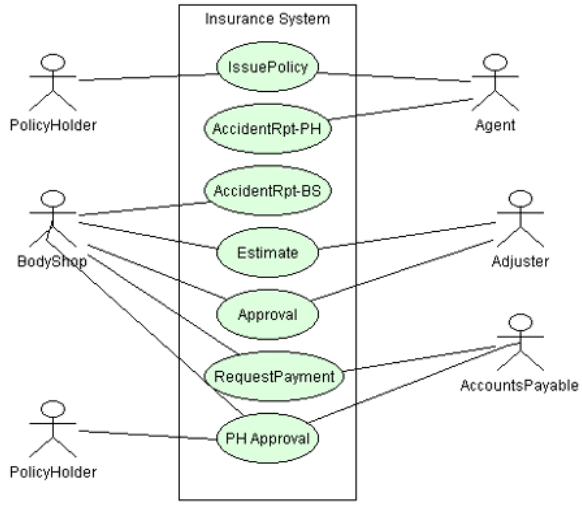
The ISO also defines standards for the requirement processes and these are detailed in ISO 9001:2000—Quality Management Systems—Requirements. These standards emphasize

- The requirement processes and the management of these processes
- The commitment and involvement of top management
- The customer focus
- The involvement of people
- Continual improvement of these processes
- The factual approach to requirement decisions

The IEEE also has developed standards for requirements. General system requirements are describe in IEEE Std. 1233-1998. Software requirements are detailed in IEEE Std. 830-1998: IEEE Recommendations for Software Requirements Specification (SRS). As stated in that document, a good SRS has the following benefits:

- Establish the basis for agreement between the customers and the suppliers on what the software product is to do.
- Reduce the development effort.
- Provide a basis for estimating costs and schedules.
- Provide a baseline for verification and validation.
- Facilitate transfer (of the software product).
- Serve as the basis for enhancement.

Figure 6.11. Use case analysis



Their IEEE document also discusses the considerations for producing a good SRS:

- Nature of the SRS
  - Functionality
  - External interfaces
  - Performance (speed, availability, response, recovery, etc.)
  - Attributes (portability, correctness, maintainability, security, etc.)
- Environment of the SRS
  - Include all needs
  - Not include design or implementation details
  - Not impose arbitrary constraints
- Characteristics of a good SRS
  - Correct
  - Unambiguous
  - Complete
  - Consistent
  - Ranked
  - Verifiable
  - Modifiable
  - Traceable
- Joint preparation of the SRS (customers, suppliers)
- SRS evolution
- Prototyping
- Embedding design in the SRS (show design constraints not design specifics)
- Embedding project requirements in the SRS (address the product not the process)

SRS templates are provided in this IEEE standard, and one such template is:

- External interface requirements
  - User Interfaces
  - Hardware interfaces
  - Software interfaces
  - Communication interfaces
- Functional requirements
  - Mode 1

Functional requirement 1.1

…

Functional requirement 1.n

…

Mode n

Performance requirements

Design constraints

Software system attributes

Other requirements

As well as general purpose software tools such as spreadsheets, databases, and group decision support tools, there are a number of specialized software tools available to facilitate different portions of the requirement processes including discovery, organization/documentation, prioritization/phasing, and change control. Discovery tools allow users to enter data about possible requirements, sort and filter requirements based on their attributes, check for possible redundancies, and disseminate requirement information to stakeholders via various paper or electronic media. Prioritization tools allow users to enter priorities and other comments about each requirement (perhaps anomalously) either in a synchronous or asynchronous manner. Phasing tools allow users to finds set of requirements that match time and/or budget constraints based upon priorities and/or other factors. Examples of specialized requirements software include DOORS/ERS by Telelogic (www.telelogic.com/products/), Analyst Pro by Goda (www.analysttool.com/), RTM by Serena (www.serena.com), SpeeDev RM (www.speedev.com), RMS by TrueREQ (www.truereq.com), Catalyst Enterprise 3 by SteelTrace (http://steeltrace.com), RequisitePro from IBM/Rational (www-306.ibm.com/software/awdtools/reqpro/), ActiveFocus from Xapware (www.xapware.com), RIQTek (http://home.riqek.com), and Cradle from 3SL (www.threesl.com/). However, these specialized tools do not fully automate the overall requirement process, they simply facilitate certain portions thereof because all of these requirement processes involve attentive and careful personal interaction amongst stakeholders.

For further details on IT requirements, see *Managing Software Requirements* (Leffingwell & Widrig, 2000), *Software Engineering Requirements* (Thayer & Dorfman, 1999), and *Mastering the Requirements Process* (Robertson & Robertson, 1999).

Chapter Summary

In this chapter, initial and overall project planning has been covered. The process of IT requirements discovery and documentation has been discussed and illustrated. Once a complete and clear set of requirements has been documented and approved by all relevant stakeholders, detail project planning can begin; such detail planning is covered in upcoming chapters.

References

Alexander, M. (2004). Software development should include security plan. Retrieved from www.adtmag.com

Davis, A., Hickey, A., & Zweig, A. (2004). Requirements management in a project management context. In P. Morris & J. Pinto (Eds.), *The Wiley guide to managing projects*. New York: Wiley.

Leffingwell, D., & Widrig, D. (2000). *Managing software requirements*. Boston: Addison-Wesley.

PMI. (2000). *The project management body of knowledge (PMBOK).* Newton Square, PA. ISBN 1-880410-22-2.

Robertson, S., & Robertson, J. (1999). *Mastering the requirements process.* Boston: Addison-Wesley.

Standish Group. (2004). Chaos chronicles. Retrieved from www.standisgroup.com

Thayer, R., & Dorfman, M. (1999). *Software engineering requirements*. New York: Wiley.