**Chapter 2. Software Project Planning**

**An urgent need can be a good motivator**. Software is usually built to address an urgent need. For the stakeholders—meaning everyone who has a concern or real interest in the success of the project—the fact that the need is urgent helps to convince the organization to dedicate time and energy to addressing it. For the software engineers, an urgent need means that their work will be appreciated and used in the foreseeable future. And for the management of the organization, it means that there is a clear direction in how they need to run things.

But urgency can also be the source of a project's most difficult problems. Urgency causes people to panic, and to potentially gloss over important parts of the project. People will rush into gathering requirements (or, even worse, rush into building the code!) without thoroughly understanding the needs of the people who want the software built.

If a project manager does not really understand the context in which the software is being built, then the only thing that the project team sees is the urgency; they lose track of the needs. They can see the individual problems that they are working to solve, but they may lose track of the big picture.

Unless the team understands the needs that drive the project, they may end up with a narrow focus, causing them to waste time addressing problems that are of little importance to the stakeholders. It's easy to build great software that solves the wrong problems, but the only way to build the appropriate software is for everyone in the project to understand and agree on both why and how that software will be built before the work begins. That's the purpose of project planning.

**Understand the Project Needs**

When a stakeholder does not feel that his needs are being met, he usually puts pressure on the project manager to provide an early version of the software, so that he can personally verify that the team really understands why the software is being built. This is a big source of communication failure between the people who need the software and the team members who are building it. When the stakeholder asks for an early version or a prototype of the software, he is usually asking for evidence that his needs are understood and being addressed. When programmers hear this request, however, they interpret it to mean that the person is interested in the details of what they are doing, and that he wants to be given a tour of the solution as it is being developed.

This may sound funny to some seasoned software engineers. Such a basic lack of communication couldn't really be that widespread...right? But most software professionals have had to sit through uncomfortable meetings where a designer or programmer gives a demo or walkthrough of all of the great work he did, only to be surprised that nobody seems to care.

The reason nontechnical people seem so bored in demos and walkthroughs is because they came to see confirmation that the technology team understands their needs. Instead, they got a lecture in software design, which they did not want. This situation can be frustrating for everyone involved: the software engineers feel unappreciated and condescended to, while the stakeholders feel like their needs are not being taken seriously.

A project that starts out like this is likely to experience scope creep, delays, and even outright failure. What's worse, nobody in the organization will really even understand why the project had so many problems. All they will see is a team that built software that didn't work properly and had to be repaired or even rewritten. This is a very bad situation for a software engineering team to end up in. Even when a team is technically proficient and capable of delivering high-quality, well-written software, when faced with a problematic project, most managers will intuitively feel that the team is incapable of delivering software without major quality problems.

What usually stalls a project is a lack of good project management. To prevent problems, the project manager must identify the people who are making decisions that affect the project and understand why they need the software built. By talking to them and writing down their needs, the project manager can set the project on its proper course—and give the stakeholders the feeling from the very beginning that the team is taking their needs seriously.

**Drive the Scope of the Project**

When the project begins, the project manager has a unique role to play. The start of the project is the time when the scope of the project is defined; only the project manager is equipped to make sure that it's defined properly. Everyone else has a role to play later on: users and stakeholders will provide expertise, requirements analysts will write specifications, programmers will build the code, etc. Everyone involved in the project has some input into the scope, but only the project manager is solely dedicated to it.

Defining the scope is the most productive thing a project manager can do to get the project underway. It is usually counterproductive to do any other project activity before everybody agrees on the scope, because that activity could fall outside of the scope without anyone realizing it. For example, many programmers will immediately begin coding proof-of-concept prototypes before even talking to the users or stakeholders; many stakeholders will encourage this because they intuitively feel that they cannot make decisions without seeing something in front of them. But building a working model is a very time-consuming way to figure out that a feature is not needed. If time is a concern, this is definitely not an efficient way to build software.

By focusing on discussing the scope and writing a vision and scope document, the project manager can ensure that the team starts out moving in the right direction. This should not be hard—when the project begins, everyone is highly suggestible. Nobody is really in her area of expertise yet: the code is not ready to be built, requirements are not known well enough to be gathered, and designs and test plans cannot even be approached yet.

Instead, there is a great deal of knowledge floating around in peoples' heads. To make sure that knowledge is captured, there must be a single person responsible for gathering it. This is the main role for the project manager at the start of the project. Once he starts talking to individual people about what they expect to see in the project and writing down their thoughts, the scope will start to coalesce and people will start feeling comfortable with the direction in which it is going. This can happen very quickly once the project manager starts asking questions and writing down the answers.

This is why the project manager is the driving force at the start of the project. It may not always feel like it, but the most important project decisions are made at the project's outset. This is when the broad strokes are laid down, when the major features of the software are defined. If there is a misunderstanding at this point, the team can be sent down an entirely wrong path, which can cause them to lose all of their momentum.

When a project team is first assembled, there is almost always a sense of anticipation and excitement among the project team and the stakeholders. The project manager can take advantage of this energy to drive the project in the right direction.

**Talk to the Main Stakeholder**

The project manager's first task in any software project is to talk to the main stakeholder . Unfortunately, it's not always immediately obvious who that stakeholder is. The project manager should find the person who will be most impacted by the project, either because he plans on using it or because he is somehow responsible for it being developed. In other words, the project manager needs to find the person who will be in trouble if the software is not developed. (There are often several people who are in this situation. The project manager should talk to each one, starting with the person who he feels will provide the most useful information.)

When a project first starts, a project manager's job is not unlike that of a tailor fitting someone for a custom suit or dress. The tailor's customers will pay a premium for tailored clothes, rather than paying less for an outfit off the rack. But this customization also means that they will need to spend time with the tailor choosing the patterns, going through fabric swatches, taking measurements, and giving some of the precise instructions necessary to customize the clothing. The customer does not see this as a chore, but rather as a perk. By giving exact specifications, the customer can get exactly what he wants. The project manager should try to form the same sort of relationship with each stakeholder that the tailor does with his customers. He can do this by working to understand exactly what it is that the stakeholder will need from the software, and then by helping the project team to deliver software that is tailored to those needs.

Unfortunately for most project managers, the typical relationship with the stakeholder is more like the relationship between a car mechanic and his customer. The customer does not see that the mechanic is using specialized skills to fix a potentially difficult problem. He just can't use his car until the mechanic says it's fixed. He wants the fix to be as fast and cheap as possible, and he doesn't fully understand why it costs so much. What's more, he's always a little suspicious that the mechanic is doing more work than necessary or ordering a part that's too expensive. The customer never knows what's wrong until the mechanic examines the car, and the cost is always an unpleasant surprise. As far as the customer is concerned, the mechanic is simply removing an annoyance, and the customer resents having to pay any money at all for it. This is exactly how many stakeholders think about their software projects and the teams that build them.

For this reason, it's important for the project manager to take responsibility for the delivery of the project from its very beginning. Each stakeholder should feel like the project manager is his main point of contact for any problem or issue with the project. He should feel that the project manager understands his needs, and that he will work to make sure that those needs are addressed in the software. He should be comfortable going to the project manager with any problems or changes at any point during the project's duration.

This sort of goodwill from a stakeholder can often be established in a single conversation or an initial meeting at the beginning of the project. The project manager can do this by blocking out time to meet with the stakeholder, leading him through a conversation about his specific needs, and then showing that he really understood those needs. The way the project manager shows the stakeholder that his needs are understood is by writing a vision and scope document , and then having the stakeholder review it.

**Write the Vision and Scope Document**

The *vision and scope* document is one of the most important tools that a project manager has; it is also one of the easiest to implement. A good vision and scope document will help a project avoid some of the costliest problems that a project can face. By writing the document and circulating it among everyone involved in the project, the project manager can ensure that each of the stakeholders and engineers shares a common understanding of the needs being addressed—and that the software must address them.

Some of the most common (and expensive) problems that a project can experience are caused by miscommunication about the basic goals of the project and about the scope of the work to be done. (The *scope* of a project usually refers to the features that will be developed and the work that will be done to implement those features. It often also includes an understanding of the features that will be excluded from the project.) By controlling the scope, the project manager can make sure that all of the software engineers' activities are directed toward building software that will fulfill the needs of the stakeholders.

The "vision" part of the vision and scope document refers to a description of the goal of the software. All software is built to fulfill needs of certain users and stakeholders. The project manager must identify those needs and write down a vision statement (a general statement describing how those needs will be filled).

Many software engineers will recognize the sinking, pit-of-the-stomach feeling when, upon seeing the software for the first time, a stakeholder or customer says, "But I needed the software to do this other thing, and I don't see that anywhere." The vision and scope document helps project managers avoid that problem by catching misunderstandings early on in the project, before they have a chance to influence the code and send the project down the wrong path.

When a project is initiated, the project manager should take the lead, talking to the stakeholders and creating a vision and scope document before the first line of code has been written. However, sometimes project managers must take over projects on which the programming has already been started. This is never an easy situation—senior managers don't usually change project managers unless the project is already in trouble. But even when a project manager has to take over a project that has already been started, it's still a good idea to go back to the stakeholders and build a new vision and scope document. This is the project manager's best chance at catching scope problems as early as possible. If the project has started to go off track, the best way to repair the damage is to assess the current needs of the stakeholders, and then to create a plan to change the software in order to meet those needs.

[Table 2-1](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch02.html#appliedprojectmgmt-CHP-2-TABLE-1) shows a typical outline for a vision and scope document. When talking to each stakeholder, the project manager should direct the conversation so that all of the topics in the vision and scope document are discussed. Typically, stakeholders will be happy to talk about all of this. They know their own needs very well, they understand who will be using the software, and they generally have many opinions about what features should go into it. The project manager should take lots of notes during this conversation—enough so that writing the vision and scope document is mostly a matter of transcribing those notes. (Stakeholders will often appreciate lots of note-taking, because it shows that the project manager is listening to them and taking what they say seriously.)

*Table 2-1. Vision and scope document outline*

|  |
| --- |
| 1. Problem Statement    1. Project background    2. Stakeholders    3. Users    4. Risks    5. Assumptions 2. Vision of the Solution    1. Vision statement    2. List of features    3. Scope of phased release (optional)    4. Features that will not be developed |

The outline itself provides a good guide for the discussion with each stakeholder. It can be used as an agenda for the meetings that the project manager uses to gather the information about stakeholder needs. It's a common mistake for a project manager to approach the writing of a vision and scope document as little more than a bureaucratic exercise. The project manager's goal should be to learn what each user, stakeholder, and project team member thinks about the software in order to develop a single, unified vision and ensure that everyone shares that vision. He should treat the document as a tool to build consensus among the stakeholders and project team members. In other words, the important Part is the discussion of the vision and scope; the document is simply a record of that discussion.

Project background

This section contains a summary of the problem that the project will solve. It should provide a brief history of the problem and an explanation of how the organization justified the decision to build software to address it. This section should cover the reasons why the problem exists, the organization's history with this problem, any previous projects that were undertaken to try to address it, and the way that the decision to begin this project was reached.

Stakeholders

This is a bulleted list of the stakeholders. Each stakeholder may be referred to by name, title, or role ("support group manager," "CTO," "senior manager"). The needs of each stakeholder are described in a few sentences.

Users

This is a bulleted list of the users. As with the stakeholders, each user can either be referred to by name or role ("support rep," "call quality auditor," "home web site user")—however, if there are many users, it is usually inefficient to try to name each one. The needs of each user are described.

Risks

This section lists any potential risks to the project. It should be generated by a project team's brainstorming session. It could include external factors that may impact the project, or issues or problems that could potentially cause project delays or raise issues. (The process for assessing and mitigating risk below can be used to generate the risks for this section.)

Assumptions

This is the list of assumptions that the stakeholders, users, or project team have made. Often, these assumptions are generated during a Wideband Delphi estimation session (see [Chapter 3](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch03.html)). If Wideband Delphi is being used, the rest of the vision and scope document should be ready before the Delphi meeting and used as the basis for estimation. The assumptions generated during the estimation kickoff meeting should then be reviewed, and any nontechnical assumptions should be copied into this section. (Technical assumptions—meaning assumptions that affect the design and development but not the scope of the project—should not be included in this document. The estimate results will still contain a record of these assumptions, but they are not useful for this particular audience.)

If Wideband Delphi is not being used to generate the assumptions, the project manager should hold a brainstorming session with the team to come up with a list of assumptions instead. (See [Chapter 3](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch03.html) for more information on assumptions.)

Vision statement

The goal of the vision statement is to describe what the project is expected to accomplish. It should explain what the purpose of the project is. This should be a compelling reason, a solid justification for spending time, money, and resources on the project. The best time to write the vision statement is after talking to the stakeholders and users and writing down their needs; by this time, a concrete understanding of the project should be starting to jell.

List of features

This section contains a list of features. A *feature* is as a cohesive area of the software that fulfills a specific need by providing a set of services or capabilities. Any software package—in fact, any engineered product—can be broken down into features. The project manager can choose the number of features in the vision and scope document by changing the level of detail or granularity of each feature, and by combining multiple features into a single one. Sometimes those features are small ("screw-top cap," "holds one liter of liquid"); sometimes they are big ("four-wheel drive," "seats seven passengers"). It is useful to describe a product in about 10 features in the vision and scope document , because this usually yields a level of complexity that most people reading it are comfortable with. Adding too many features will overwhelm most readers.

Each feature should be listed in a separate paragraph or bullet point. It should be given a name, followed by a description of the functionality that it provides. This description does not need to be detailed; it can simply be a few sentences that give a general explanation of the feature. However, if there is more information that a stakeholder or project team member feels should be included, it is important to include that information. For example, it is sometimes useful to include a use case (see [Chapter 6](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch06.html)), as long as it is written in such a way that all of the stakeholders can read and understand it.

Scope of phased release *(optional)*

Sometimes software projects are released in phases: a version of the software with some subset of the features is released first, and a newer, more complete version is released later. This section describes the plan for a phased release, if that approach is to be taken.

This is useful when there is an important deadline for the software, but developing the entire software project by that deadline would be unrealistic. The most common way to compromise on this release date is to divide the features into two or more releases. In that case, this section should identify specifically when those versions will be released, and which features will be included in each version. It's reasonable to divide one feature up between two releases, as long as it is made clear exactly how that will happen.

If a project manager needs to release a project in phases, it is critical that the project team be consulted. Some features are much more difficult to divide than others, and the engineers might see dependencies between features that are not clear to the stakeholders and project manager. After the phased release plan is written down and agreed upon, the project team should always be asked to re-estimate the effort and a new project plan should be generated (see below). This will ensure that the phased release is feasible and compatible with the organization's priorities.

Features that will not be developed

Features are often left out of a project on purpose. When a feature is explicitly left out of the software, it should be added to this section to tell the reader that a decision was made to exclude it. For example, one way to handle an unrealistic deadline is by removing one or more features from the software, in which case the removed features should be moved into this section. The reason these features should be moved rather than deleted from the document is that otherwise, readers might assume that they were overlooked and bring them up in a review. This is especially important during the review of the document because it allows everyone to agree on the exclusion of the feature (or object to it).

**REVIEW THE VISION AND SCOPE DOCUMENT**

Once the vision and scope document has been written, it should be reviewed by every stakeholder, by the members of the project team, and, ideally, by at least a few people who will actually be using the software (if they are available). Performing this review can be as simple as emailing the document around and asking for comments. The document can also be inspected (see [Chapter 5](https://learning.oreilly.com/library/view/applied-software-project/0596009488/ch05.html)). However the document is reviewed, it is important that the project manager follow up with each individual person and work to understand any issues that the reviewer brings up. The project manager should make sure that everyone agrees that the final document really reflects the needs of the stakeholders and the users, and that if they build the software described in the second half of the document, then all of the needs in the first half will be met. Once the document has been reviewed and everyone agrees that it is complete, the team is unified toward a single goal and the project can be planned.

**SOFTWARE PROJECT  
PLANNING**

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_fig0002.jpg

*Nobody plans to fail –  
they just fail to plan.*  
– **Anonymous**

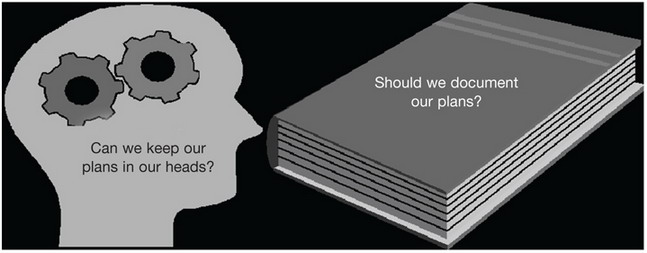
*If I were given six hours to fell a tree,  
I would spend the first four hours sharpening the axe.*  
– **Abraham Lincoln**

**INTRODUCTION**

Most articles and books about achieving success (in any endeavor) begin by describing the necessity to plan and to plan well. In rare cases, success can be achieved without planning, but planning reduces the risk of failure and increases the chances of success. Better yet, planning coupled with control (control from the point of view of project discipline, including measuring progress and taking corrective actions) brings more predictability to the probable outcome of an endeavor.

An often-asked question is, “Can I plan in my head or should planning be documented on paper?” Although planning is a necessity, documenting the plan (or *planning on paper*) is not always a necessity. For instance, for a small, short-duration endeavor, mental planning may be adequate ([Figure 5.1](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_05.xhtml#fig0009)).

Few of us actually omit planning. Usually, we conduct the planning — it’s just the degree of rigor with which we plan (including documenting the planning activities) that is open to discussion.



**Figure 5.1.** The planning dilemma.

Planning on paper has advantages. A documented plan can:

•Be *reviewed by others* to see if any important aspect has been overlooked, thereby improving the plan (or self-reviewed “after the dust has settled”)

•Act as a *point of reference* for stakeholders concerned with or involved in the project

•Facilitate *control and performance evaluation* during execution

•Facilitate *validation of the planning parameters* by providing a baseline for comparison of the actual values generated during execution

Except for very small projects, written documentation of a plan is a good idea.

Once the decision has been made to document planning, the next question is usually, “What level of granularity should be used?” The granularity of planning (or the required detail) depends on:

•*Duration* of the endeavor

•*Number of resources* employed

•*Complexity* involved

•*Relationship* between the duration, number of resources, and complexity

•*Geography* of the project

Now consider some aspects of the duration of the endeavor, the number of resources employed, and the complexity involved in the project:

•The longer the *duration*, the greater the necessity for rigor. (If a project has no time constraints on completion, the level of rigor and granularity can be reduced. In the real world, however, it is *duration* that is often constrained, which increases the need for planning rigor.)

•As the *number of resources* employed in a project increases, the level of planning rigor increases.

•As *complexity* (of all varieties and above what is normal) for the team members increases, the greater is the need for greater planning rigor.

•Different combinations of *duration*, the *number of resources* employed, and *complexity* require different levels of rigor in planning.

Numerous other questions could be asked about project planning, but before proceeding with any further discussion on planning, let’s define planning.

**PLANNING DEFINED**

Planning is defined as *the intelligent estimate of resources required to perform a predefined project successfully at a future date within a defined environment*. This definition of planning contains several key terms:

•*Estimate* indicates that planning is preceding performance and that it is based on organizational norms (also known as organization baselines). Simply, an estimate is a prediction of the future.

•*Resources* are the four M’s of *men* (human resources, either male or female), *materials*, *methods*, and *machines* (equipment). Resources are always applied over a period of time (duration).

•*Project* indicates a specific scope of work that can be defined as a project (see [Chapter 1](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_01.xhtml)).

•*At a future date* indicates that the dates for executing the project are in the future and are typically decided during the course of planning.

•*Within a defined environment* refers to the environment in which the work will be performed. The environment is either known or defined during the planning exercise. Any variation in the environment will have an effect on the plan. *Environment* may also refer to the technical environment, work logistics, workstation design, processes and methods of management, prevailing morale in the workplace, and the corporate culture to name a few.

The definition of planning also provides a framework for evaluating the process of planning.

In several aspects, the planning for software development projects is the same as the planning for many other types of projects. In software development planning, however, the planning process is tailored to suit the specific attributes of software development. So, what are the attributes that make software development projects unique?

•*Output is not physical*: The output from a software development project is not physical — in the sense that the primary deliverable is not an actual physical component. *Functionality* is the primary output of a software development project. (Physical and nonphysical resources are consumed, however, when a software product is created.)

•*Process inspection does not facilitate progress assessment*: In a manufacturing organization, the conversion of raw materials into work-in-progress and finally into finished goods is proof of progress. (Some say that progress may be assessed by the noise made by manufacturing equipment.) In a software development organization, however, visual inspection is not enough to ensure that work is being done and progress is being made. In a software project, *functional* software is the only true marker of progress.

•*Software engineering tools have limited predictability***:** Although significant progress has been made in software engineering tools, these tools do not have nearly as much precision as engineering drawings and cannot produce the predictability that is seen in other engineering disciplines. Much of the energy expended in software development projects continues to be “sweat equity.”

•*Professional associations in the software development field lack practice and behavior standards:* Organizations such as the Institute of Electrical and Electronic Engineers (IEEE) have defined some standards, but these standards do not rise to the levels of specificity and granularity that are found in other engineering fields.

•*Productivity and quality are dependent on human beings*: Significant improvement has been made in software development; however, for productivity and quality, the processes used continue to be largely dependent on human beings. Tools are available to support development and testing, but to meet the standards found in other engineering disciplines these tools need to evolve further. The goal must be shifting the onus for productivity and quality from human beings to tools in the hands of humans.

Because software development continues to rely primarily on human endeavors, the rigor of planning needed becomes even more significant than that required for engineering projects (an environment in which tools provide a major impetus to the process). For example, in some engineering projects, a simple schedule based on PERT/CPM will suffice, whereas software development projects (especially large ones) require increased rigor and planning. The plans typically required for a software development project are now described in subsequent sections, but first, let’s review the general attributes of a software project:

•A project has a definite beginning and a definite end.

•The project deliverable is software and the related artifacts (e.g., documentation).

•The activities in a software project may include defining the user and software requirements, software design, software construction, software testing, acceptance testing, and software delivery, deployment, and handover.

Project selection, acquisition, and post-handover activities are not part of a software development project.

**PLANS PREPARED IN SOFTWARE PROJECT MANAGEMENT**

A common misunderstanding among members of the software development fraternity is that a *schedule* constitutes software project planning. This is categorically untrue. Software project planning goes far beyond scheduling. Several plans are typically prepared for a large software development project.

***A project management plan.*** A PMP describes how a software project will be managed. In engineering projects, how a project will be managed is covered in the standard operating procedures/policies (or SOPs) of an organization’s production environment or production facility. SOPs work well for engineering projects because all projects are managed similarly. Therefore, a completely new management plan for every project may be unnecessary. The SOPs also ensure that how a project is to be managed is well understood. The software project developmental environment, however, is much more dynamic. In software projects, the developmental environment is completely different for almost every project, which necessitates the need to plan and document how each project will be managed. Information contained in a PMP includes:

•Project demographic information

•A software estimate (software size, effort, cost, and schedule)

•Milestones and delivery schedules

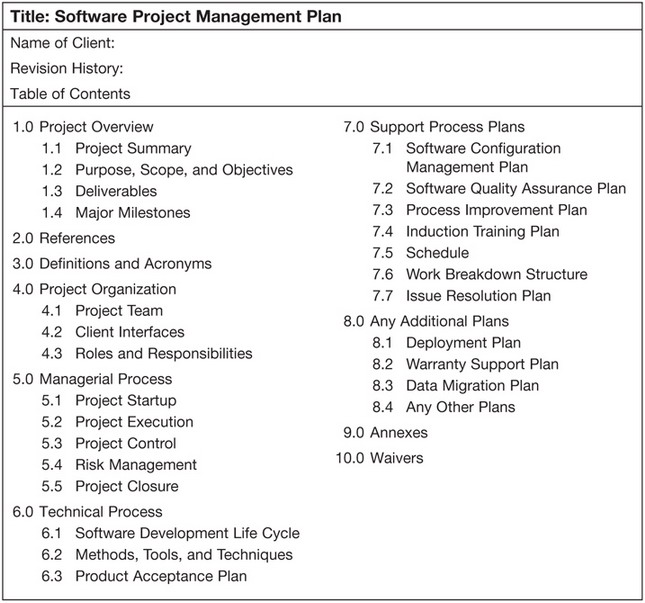
•Delivery acceptance criteria

•Human resources requirements and a projected timeframe for when they will be required

•Management methods (including but not limited to work allocation and management, information and source code management, quality control, communication management, etc.)

•Tools to be used for the project (development tools, testing configuration tools, and project management tools, etc.)

The elements of a software PMP template are shown in [Figure 5.2](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_05.xhtml#fig0010). IEEE Standard 1058 provides details for a PMP and a suggested template for documenting a plan.



**Figure 5.2.** Software project management elements.

***A configuration management plan.*** A CMP describes how code and noncode assets of a project will be managed. Information in a CMP includes:

•Naming conventions to be followed for project artifacts, including documents and code units of all types (including databases and tables); procedures for managing changes to configuration items

•Organization of project information to facilitate access by project teams when a reference is needed

•References to organizational standards and processes for use in the project

•Code and code library organization, check-in and check-out criteria, authorizations, and procedures for state changes of source code artifacts (from development to review/testing, to integration and delivery, etc.)

IEEE Standard 828 provides details of a CMP and a suggested CMP template.

***A quality assurance plan.*** A QAP describes how a project will ensure that the deliverables meet the quality requirements for the project. Information in a QAP typically includes:

•Standards selected for use during the project (coding, design, and testing guidelines)

•Quality control activities proposed for the project (code walk-through, review of requirements and design, proposed tests including but not limited to unit testing, integration testing, functional testing, negative testing, end-to-end testing, system testing, and acceptance testing)

•Software metrics to be collected for the project and how they will be used

•Processes, procedures, and events that trigger the need for causal analysis, whether for failures, defects, or even successes

•Audits proposed for the project and who will perform them

IEEE Standard 730 gives details of preparing a QAP and a suggested QAP template.

***A schedule.*** The schedule contains a work breakdown structure for the project, including the start and end dates and the resources required for each of the activities. The schedule document is used to plan and to monitor the progress of the project. Analysis techniques, such the Critical Path Method and the Program Evaluation and Review Technique (often referred to collectively as PERT/CPM), are useful to evaluate task flows and relationships. The CPM is a step-by-step project management planning technique that identifies the tasks in a project that are on a *critical path*. Tasks (activities from the start milestone to the end milestone and in between) that are on a *critical path* are those tasks that are critical in meeting the project’s schedule and for the success of the project. The goal of identifying the critical tasks is to prevent time-related problems. Knowledge gained from the CMP is used to focus attention on the truly important tasks so that a project’s overall completion date is not impacted. (*Note*: Because the critical path shifts over time as project execution progresses, the critical path should be monitored so that events do not suddenly become overwhelming.) PERT/CPM can also be used to assist in resource allocation. Through the use of probability theory, PERT helps a project manager to understand and mitigate the uncertainty that is inherent in working out a schedule. Knowledge of PERT/CPM techniques is essential to arriving at a credible schedule for a project. (Many resources describing how to perform PERT and CPM analyses are available on the Internet.)

***An induction training plan.*** Also known as an *onboarding* plan (as well as an initiation or assimilation plan, etc.), an induction training plan contains the training requirements for new team members who join the project. Explicitly stated are the requirements for bringing a new team member up to par with other team members. Typically, the induction training plan focuses on processes and standards rather than on explicit technical requirements, which have a longer-term learning horizon). Topics also included are knowledge building about the project plans that are being leveraged on the project: the “how’s” of executing and controlling the project; the quality assurance activities; the mechanisms for communication and issue resolution; project-specific tool/development platform training, etc.

***A build plan.*** The strategy for building the code is included in a build plan. This plan details the build period (ranging from continuous to daily or some other period) and how the build will be tested and validated. Also included is the number of planned builds for delivery of the product to the client.

***A deployment plan.*** Contained in a deployment plan is a description of the target location of the project’s functionality, including deployment of the hardware, the system software, the middleware, and pilot runs.

***A user training plan.*** A user training plan outlines the user training, the deployment strategy (classroom, online, etc.), and the duration and includes an anticipated schedule for the training.

***A handover plan***. Details included in a handover plan describe how the system will be handed over to the team that will operate the system. Also included in the plan are the handover timelines, the team or the person who will accept the application, the artifacts required at handover, acceptance test criteria, and any required signoffs.

***A software maintenance plan.*** The mechanisms for identifying and prioritizing maintenance work requests, any required service levels for maintenance, and the support turnaround times necessary for maintaining the software are included in a software maintenance plan.

All of these plans may not be required as separate documents (depending on the methodology used, the content also may not be applicable). For instance, in a smaller project, the plans described above can easily be included in the PMP. In a medium-sized project, usually a PMP, CMP, QAP, and a schedule are prepared (any other plans are included in the PMP).

Next, how to carry out software project planning and the preparation of typical project planning deliverables are described in greater detail.

**THE PROJECT MANAGEMENT PLAN**

A PMP is the *top-level plan* that consolidates all of the relevant information about a project, from the purchase order to the initial estimates and requirements, into the plan. The PMP also incorporates the methods that will be used for managing the project, the project management tools to be used, the project milestones, the communication protocols, and the mechanisms for escalation and issue resolution.

**Resources**

A resource plan is typically a subsection of a PMP. Once a project has been estimated, a work schedule for execution of the project can be developed. The work schedule provides details of the resources required and the dates when the resources will be needed.[1](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_05.xhtml#ref1) Attributes that influence the human resources aspects of estimation include:

•*Skill sets required for the project*: The initial, required skill sets are derived from the technical specifications of the project. (Over the life of a project, innovation may alter the skill sets needed.)

•*Size of software to be developed*: The size of the software is estimated using accepted software size measures, such as Function Point Analysis, The Netherlands Software Metrics Association (NESMA) Function Point Analysis, or Software Size Units (SSU).

•*Amount of effort required to deliver the project*: Effort is directly estimated using estimation techniques, such as parametric estimation, task-based estimation, analogy, or Delphi estimation. Effort also can be derived from the estimated software size. (Remember from an earlier discussion that effort and duration are not the same.)

•*Duration that resources will be required on the project*: Duration is estimated by allocating the resources and assigning calendar dates to the activities that have to be performed to execute the project — a process called *scheduling*. (Scheduling will be covered in greater detail in [Chapter 9](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_09.xhtml)).

•*Likely dates for resources*: Based on estimated and compiled data, the dates on which the resources are likely to be needed by the project are derived. Estimates and schedules are fluid and can be adversely affected by events; therefore, monitoring is critical. A schedule is not a “fire and forget” task.

**Skill Sets**

Capers Jones, a leading authority in software estimating, has said that IT has more specialties than any other profession. In a large project, therefore, a wide range of different skill sets may be needed. In addition to a project leader/SPM, several roles are typical:

•*Programmers*: to write the necessary programs

•*Database administrator/database specialists*: for data modeling and to design the database, to develop stored procedures (programs written at the database end for data handling), and then to offer assistance to programmers in the efficient development of data handling routines (DBAs can also carry out data migration or assist the team in data migration activity as applicable.)

•*Team leaders*: to lead and manage teams of programmers, testers, and DBAs on a day-to-day basis

•*Software testers/testing specialists*: to prepare test plans and test cases, carry out software testing, and guide the team to ensure that testing is properly carried out and all defects are uncovered and rectified

•*Language smiths*: to assist in troubleshooting programming issues (Language smiths, also known as lead programmers or expert programmers, are experts in the programming languages used on a project. Language smiths use their expertise to assist in troubleshooting programming issues and are often requested on an “as required” basis.)

•*Software (solution) architects*: for process modeling to develop application architecture and to integrate the developed solution

•*Business (systems) analysts*: to interact with the customer(s) to understand requirements and to translate those needs/requirements into a form which can be understood and used by the software developers to produce the solution that meets the client requirements (In some cases, business analysts also act as a proxy for the customer within the project team.)

•*Configuration controller*: to ensure that the artifacts (both information and software) are available to various team members and to ensure that deliveries to the customer contain the correct versions of all deliverables (also known as a configuration manager)

•*Process coordinator*: to ensure that the organization’s processes are implemented and that process-related information is made available on time to concerned functionaries of the organization

Other possible roles are process and product quality assurance analysts (PPQA), user interface designers (UI), usability testers, etc. The list could go on and on.

At times, some roles are handled on a part-time or as-needed basis. A project leader or project manager often takes on the roles of configuration controller, process coordinator, and software architect. Programmers may take on the role of testers. Once the required skill sets and the duration for which they are required are identified, resource requests can be placed with the department that allocates human resources to projects.

**Computer Systems**

Depending on the nature of the project, a project team needs various hardware items for execution of a project execution. Typical hardware and system software requirements for a project include:

•Special computers (based on project needs)

•Personal computers (with appropriate terminal emulation software, if necessary, to connect to the development machine/server or appropriate system software, a development tool kit, and any other necessary tools)

•Networking hardware and software

•Connectivity to customer machines (if the project is to be executed from a remote location)

•Bandwidth (if communication with a remote customer or testing of a Web application is involved)

•Special software (databases, programming languages, testing tools, configuration management tools, documentation tools, team collaboration tools, etc.)

**Project Management Method**

A number of methods may be used to manage a project. (*Note*: Methods are part of the *Project Management Body of Knowledge* that forms the basis of PMP© certification.[2](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_05.xhtml#ref2)) Project management methods include work allocation, progress measurement and review, and communication.

***Work allocation.*** In the work allocation method, work is allocated to the human resources to execute the *tasks*. Then progress reporting is done at the *task level*. Work allocation may use a variety of vehicles: an Excel sheet, Microsoft Project or Primavera scheduling tools, or PMPal, a work breakdown structure (WBS) collaboration tool. Work allocation may also be performed using a tool such as Microsoft Project Server, in a formal email, and by telephone or even in person. Reporting then leverages the same tool or method.

***Progress measurement and review.*** In progress measurement and review, tools and methods are used to ensure that the status of a project is clearly understood. Progress measurement and review techniques include earned value analysis (EVA) and line of balance (LOB). A weekly status report is the most common progress reporting vehicle for projects. (*Note*: In agile projects, a daily stand-up meeting and monthly sprint reviews/planning sessions take the place of status reports.) In all cases, from EVA to stand-up meetings, the reporting process is used as a basis for progress review and for deciding on action points.

***Communication.*** Meetings, emails, and telephone calls are the most frequently used mechanisms for communication within and outside a project team — albeit new tools, such as wikis, Twitter, and instant messaging, are in the process of supplementing email and telephone messaging systems due to their intimacy and immediacy. The communication plan in the PMP should cover several scenarios:

•Communicating within the project team

•Communicating work allocation and completion dates

•Progress reporting

•Communicating with the client

•Communicating with project support groups

Other considerations related to communication concern the environment and issue-resolution and escalation mechanisms:

•*Environment*: Ensure that the tools, techniques, hardware, system software, database, integrated (also interactive) development environment (IDE), testing tools, CM tools, and folder structures for artifacts in various states that are required for the project are clearly described.

•*Issue resolution*: Ensure that the process is described so that whenever there is ambiguity and clarifications are needed that an issue is raised and tracked to resolution. Description of the process must include the mechanism used to record all issues for the project, how the issue is communicated to the appropriate person, and how the issue is tracked to resolution. ([Appendix E](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_APPENDIX_E.xhtml) discusses issue-resolution mechanisms in greater detail.)

•*Escalation*: Ensure that the process to raise an issue to the next higher level is described. Include the levels to which an issue can be escalated, when to escalate an issue, and to whom the issue should be escalated.

Typically, issue-resolution and escalation mechanisms form part of a PMP.

**THE CONFIGURATION MANAGEMENT PLAN**

A software development project has several configurations:

***Development.*** The development configuration is the arrangement of the hardware (the development machines: PCs, servers, and networks) and software (the development platform, including the programming language(s), database(s), IDE, and third-party software used in the project) to be used by the programmers for developing the software product. A development configuration plan typically has two distinct parts:

•*Code*: the source code being developed

•*Information/documentation*: information received from the client; developed for use in the project (requirement specifications, design, test plans, test cases, etc.); and generated by the project (test and review logs); all change requests

***Review/testing.*** The review/testing configuration is the arrangement of the hardware and software to be used by the reviewers and testers. Generally, software does not change in the review/testing configuration. Programs enter the configuration and after testing:

•Are returned to the development configuration for rectification *or*

•Are promoted to the integration configuration for integration with other code units

Software is transient when in the review/testing configuration, i.e., the data portion of the development configuration is used for testing and to unearth any defects in the software.

***Integration (build).*** The integration/build configuration is the arrangement of the hardware and software to be used by product integrators. Integration is the process that receives software components and integrates them into the build of the product. Software components enter this configuration only when they have been reviewed, tested, and all known defects have been satisfactorily rectified (or put on a backlog).

***Delivery.*** The delivery configuration is the arrangement for delivery of the software components to the client. Typically the delivery configuration contains some combination of the following:

•The software build

•The source code components

•Third-party software

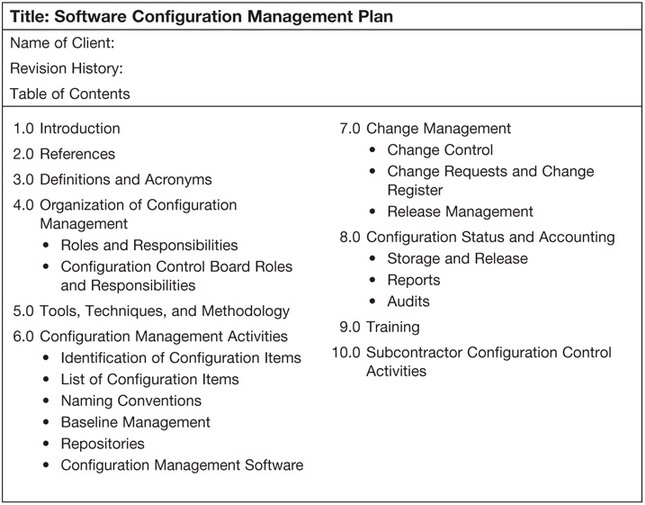
•Software libraries

•Artifacts received from client

•Images

•User documentation and training materials

•Installation guide, operations guide, and troubleshooting manual



**Figure 5.3.** Configuration management plan elements.

***Deployment (target/production).*** The deployment configuration is the arrangement of the hardware and software components in the target system on which the developed software will be deployed and used.

All required configurations are determined and then documented in configuration management planning. Because different configurations are needed at different levels of granularity at various times in a project, plans will be modified as needed. As the level of knowledge changes, definitions should be also augmented. (The necessity to modify plans and to update the project definitions is why a plan is said to be “a living and breathing document.”)

Daily configuration management activities generally revolve around two basic roles: moving components from one configuration to another and the processes required to ensure that the “right” versions of components (software and information artifacts) are assembled for delivery to the customer. Suggested template elements for a CMP are shown in [Figure 5.3](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_05.xhtml#fig0011).

***Obsolete artifacts.*** An important aspect of configuration management planning is the management of artifacts that are undergoing improvement and the management of obsolete artifacts that are created by every project. Obsolete artifacts are not destroyed — at least not until a project is complete. A mechanism is needed to ensure that concerned parties refer only to the appropriate set. (The word *appropriate* is used because at times previous baselines are needed as much as an unambiguous reference to the “current set.”) A simple approach to managing artifacts is to create three folders:

•*Current*: all relevant, active artifacts that are complete and which will be referred to when needed

•*Archives*: all previous versions of artifacts

•*In process*: all artifacts that are being developed/revised

Ensure that a version of an artifact is in *only one* of the folders. No artifact of the same version is to be duplicated (except as a project backup).

**Naming Conventions**

Naming conventions are typically part of a CMP or part of an organizational standard referred in a CMP or PMP. Why are naming conventions needed? Naming conventions:

•Prevent duplicate names or bring clarity when similar names are used

•Easily recognize the contents of the artifact

•Easily identify a group of artifacts (such as all artifacts related to a specific module)

•Achieve uniformity in the naming of artifacts across teams in the same project and across different projects within the organization

In programming, naming conventions allow one type of variable to be distinguished from another type of variable (e.g., programming variables and table fields).

One approach to naming conventions is to use prefixes to distinguish between the various categories. Several prefixes can be combined to provide a rich layering of meaning. Typical naming conventions include:

•Document names

•Program/subprogram names

•Screens, reports

•Numeric variables

•Alphanumeric variables

•Flags

•Counters

•Database table names

•Database table field names

•Error messages

•Information messages

•Window

•Controls (combo box, text box, command buttons)

**Change Management**

Change is an inevitable occurrence in a software project (similar to death and taxes!). Identifying (or defining) the process required for change management typically occurs in a configuration management plan. (See [Chapter 8](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_08.xhtml) for more detailed information about change management.) Inclusion of the contents of the change management process in the CMP is not an absolute, but no matter where change management is documented, the contents of the process are important. Change management in software projects includes:

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpg*Receiving change requests*: designates a single point for receiving CRs (from all sources), for consolidating the CRs, and for maintaining a change register (also known as a change log)

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpg*Analyzing change requests*: specifies who is responsible for analyzing CRs (Analysis includes developing an understanding of the impact on schedule, effort, and cost.)

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpg*Establishing a change control board*: receives data from the CR analysis process and uses that data to make accept or reject decisions or to request more information about the CR. If the CR is accepted for implementation, the change control board:

•Decides when to implement the CR (as and when received, in a later release, or to retrofit all CRs in the final release?)

•Decides how to absorb the impact of the CR (internally or pass impact on to the customer?)

•Obtains/accords approval for implementation of the CR

•Implements the CR

•Monitors quality control of CR implementation

•Closes the CR

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpg*Reporting progress*: determines the mechanisms to be used to track all CRs received, to track all CRs to resolution, and to communicate the status of all CRs to all concerned parties on a periodic basis

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpg*Closing change requests*: closes CRs when no further attention is required (Authority for closing a CR rests with the change control board. When a CR is closed, the requesting party receives notification of the final disposition of the CR.)

**THE QUALITY ASSURANCE PLAN**

Quality assurance planning focuses on achieving the specified level of quality of the artifacts to be produced by a development team. A QAP usually contains:

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpgStandards to be used in the project:

•Coding

•Database design

•GUI design

•Test case design

•Testing

•Review

•Organizational process reference

•Other organization-specific standards

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpgQuality goals for the project:

•Defect injection rate

•Defect density

•Productivity for the project’s artifacts

•Schedule variances

•Other project-specific quality goals

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpgQuality assurance and control activities to be implemented in the project:

•Code walk-through

•Peer review

•Managerial review

•Types of tests to be carried out during project execution (At a minimum, testing should include unit, integration, system, and acceptance testing.)

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpgMeasures and the processes for measurement (Cover the defined quality levels, the periodicity of testing, and the reporting mechanisms.)

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpgCausal analysis (process and schedule) for positive and negative variances

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpgSchedules for proposed project audits:

•Periodic conformance

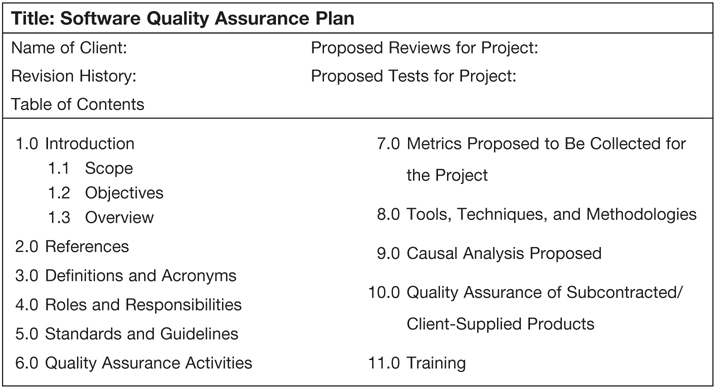
•Phase-end

•Criteria for investigative audits

•Delivery

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpgProcess improvement activities (if any)

https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/images/9781604270341_chr0001.jpgProgress reporting mechanisms for the status of quality assurance activities implemented in the project (for all concerned parties)



**Figure 5.4.** Quality assurance plan elements.

A suggested template for a QAP is shown in [Figure 5.4](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_05.xhtml#fig0012).

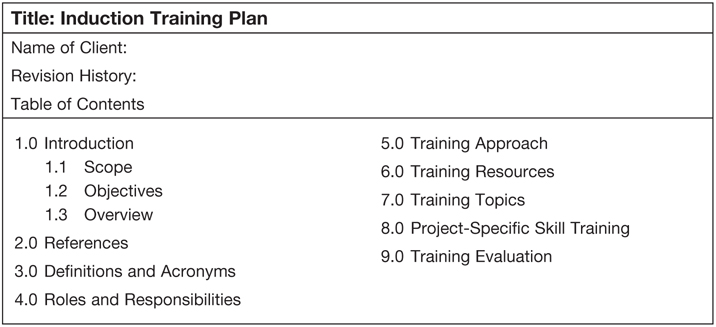
As with all other plans, each section of a QAP must be evaluated for its pertinence to the project. For example, the section on standards could contain a reference to existing organizational standards. (*Note*: The authors are firm believers in doing what is needed — not just approaching project quality management and control as a rote checklist.)

**THE SCHEDULE PLAN**

Scheduling planning is best achieved by using scheduling software (e.g., Microsoft Project or Primavera). All of the activities that are needed to execute a project are enumerated; their predecessor relationships are defined; the resources are allocated; and the dates are set for the activities. ([Chapter 9](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_09.xhtml)provides greater detail about scheduling a project.)

**THE INDUCTION TRAINING PLAN**

An induction training plan (also known as an *onboarding* plan) describes how personnel are to be brought up to speed to ensure the highest level of efficiency for the project throughout its entire life cycle. An induction training plan contains the training topics, duration of training, and the possible faculty for each topic needed by the personnel before beginning to work on the project. The plan should also include a waiver process for the personnel who do not require training due to previous training or their level of experience. Training topics may include:



**Figure 5.5.** Induction training plan elements.

•Project plans

•Team communication methods

•Quality assurance activities

•Issue-resolution mechanisms

•Escalation procedures

•Development platform

•Training methods

•Availability of self-study materials

•Waiver process

Suggested template elements for an induction training plan are shown in [Figure 5.5](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_05.xhtml#fig0013).

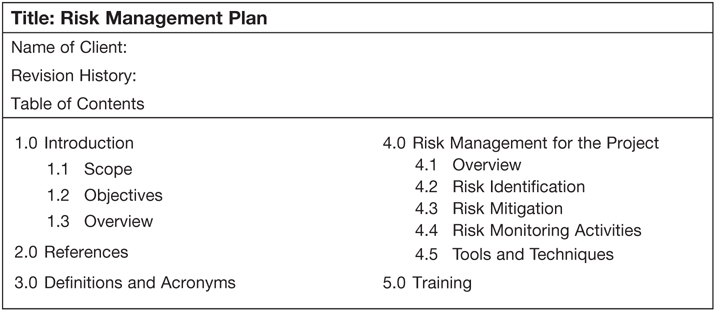
**THE RISK MANAGEMENT PLAN**

A risk management plan describes how risks will be identified, prioritized, and managed across the life of the project. Having a risk management plan helps to ensure that risks do not disrupt progress if at all possible. The risk management plan may be included as a part of a PMP or developed and documented as a standalone plan. Typical risk management activities include:

•Risk identification

•Risk quantification

•Risk prioritization



**Figure 5.6.** Risk management plan elements.

•Risk mitigation

•Risk monitoring and reporting

Suggested template elements for a risk management plan are illustrated in [Figure 5.6](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_05.xhtml#fig0014).

**THE BUILD PLAN**

A build plan contains the strategy for building the software project’s code and details the build period (ranging from continuous to daily or to some other period) and how the build will be tested. Details of the build plan include when and how functionality will be delivered to the client. A typical build plan contains:

•The approach for integration (i.e., top-down or bottom-up)

•Roles and responsibilities for preparing the builds

•Configuration of the integration environment

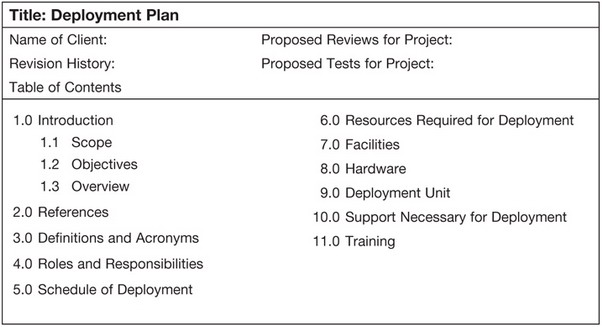
•Quality assurance activities before accepting components into the build environment

•Quality assurance activities after integrating each component, after integrating each module, and at completion of the build

**THE DEPLOYMENT PLAN**

A deployment plan contains a description of the target location of the project’s functionality, including the deployment of hardware, system software, middleware, and pilot runs. A deployment plan typically contains:

•A schematic diagram of the deployment components, including hardware, software, networking, etc.



**Figure 5.7.** Deployment plan elements.

•Floor plans for deployment of hardware and networking (if necessary)

•A bill of materials (lists all components being deployed along with the technical specifications of each of the components)

•Quality assurance activities planned for deployment

•Technical methods for deploying the configuration (if necessary)

Suggested template elements for a deployment plan are shown in [Figure 5.7](https://learning.oreilly.com/library/view/mastering-software-project/9781604276909/9781604270341_ch_05.xhtml#fig0015).

**THE USER TRAINING PLAN**

A user training plan describes how users of the system will be taught to use the functionality being delivered. A user training plan minimally contains:

•A delineation of the types of users to be trained and the topics for each type of user

•Details of each of the training topics

•Course material for each course (including, but not limited to, training slides, teaching notes, lesson plans, session breakdown, and participant handouts)

•A schedule of the courses to be conducted

•Details of the facilities needed for conducting the training

**THE HANDOVER PLAN**

A handover plan describes how the functionality will be delivered to the client or support organization. A typical handover plan contains:

•A bill of materials (all components to be handed over to the client)

•The mode of handover/takeover

•The required sign-off details

•A schedule for the handover

**THE SOFTWARE MAINTENANCE PLAN**

A software maintenance plan is typically driven by contractual requirements. At a minimum, however, a software maintenance plan describes the activities, roles, and processes for the usual warranty period (sometimes, longer warranty periods may be requested; if so, a separate software maintenance project is usually spawned). A software maintenance plan may contain:

•The process for raising requests for software maintenance

•Formats and templates for raising software maintenance requests

•Service level agreements (including turnaround times for software maintenance requests)

•The procedure for classifying software maintenance requests and prioritizing them

•Issue-resolution mechanisms and escalation mechanisms during maintenance

•The environment for software maintenance

**THE DOCUMENTATION PLAN**

Documenting of software project plans differs significantly from standard business writing. Software project plans are documents that are used by many individuals as a reference for guiding human efforts and for incurring expenditures. Therefore, approach the writing of a software project plan document as if it were an engineering drawing. Attributes of an engineering drawing include:

•*Unambiguous representation*: The same inference would be drawn from the document irrespective of the person who is interpreting it.

•*One fact — one location*: A fact is presented at one *and only one* location and is never repeated. Presentation of information at multiple locations may cause conflict or create a maintenance nightmare.

•*Specific language*: No free-flowing language is used in writing project plans. Project plans are *not* literature.

Therefore, using the analogy of an engineering drawing, a project plan should:

•Adhere to the documentation guidelines of the organization

•Avoid duplication of information at multiple locations

•Avoid ambiguity

Keeping in mind that multiple individuals are likely to prepare plan documents, uniformity can be achieved in an organization through the use of templates and reviews. Every organization, therefore, should define its templates. One suggestion is to start by using templates from industry associations, standards organizations (e.g., IEEE), or a consulting group.

**ROLES IN PLANNING**

Collaboration between various groups within an organization is critical in achieving effective project planning. At least two entities in an organization impact project planning: the organization that provides the infrastructure and the individual who carries out the project planning.

**The Organization**

To plot a project’s future, an organization needs project planning. Therefore, to facilitate process planning, the organization provides an infrastructure that facilitates and enables effective project planning:

•Development, establishment, implementation, and continuous improvement of the project planning process in the organization (procedures, templates, formats, and quality assurance for plans)

•Implementation guidelines and standards (documentation guidelines, checklists for the preparation and review of plans, and estimation guidelines and productivity figures for various technologies used in the organization)

•Establishment of a PMO (or similar) that takes charge of all project plans and assists all concerned individuals in preparing project plans and that also acts a lightening rod to receive feedback and to ensure that feedback is analyzed, acted upon, and incorporated into processes, standards, and guidelines

•Arranging for peer and managerial reviews of all plans at the preparation stage and, upon completion of a project, conducting a variance analysis to capture the best and worst practices and to measure the efficacy of the project plans

•Development and population of a knowledge repository for project planning that acts as the corporate “memory” of past estimates and project plans (a repository) so information can be used as reference for project planning

•Providing structured training for planning projects

•Recognizing that project planning is a specialist activity and subjecting it to the rigors of process improvement

•Rewarding individuals who excel at project planning

**The Software Project Manager**

Individuals can “make or break” project planning. An individual who is vested with the responsibility of project planning should be a person who strives to excel at project planning. In addition to making the best use of the available infrastructure, an SPM who is well versed in project planning can achieve effective planning for the projects. A good project planner can add value to the organizational project planning process through:

•Diligently planning the project and preparing plan documents so that they adhere to organizational processes, standards, and guidelines

•Recognizing that project planning is important. It is not just preparing documents — it is *planning the project*. The documents are an offshoot of the planning process that are to be used for the purposes of review, improvement, and reference by all concerned groups in the organization

•Assisting the organization in developing, establishing, implementing, and continuously improving the project planning process

•Adhering to organizational processes, standards, and guidelines in “letter and spirit”

•Giving feedback to concerned parties

•Participating in process improvement activities wholeheartedly

•Carrying out the project planning activity to the best of one’s ability as diligently as possible

Individuals who are poorly suited for project planning usually whine about the planning process. They generally cause more effort to be spent on the process than needed. Vesting the exercise of project planning in an SPM or a team member who is unsuited for project planning is just one of many potential pitfalls in software project planning, the subject of the next section.

**PITFALLS IN SOFTWARE PROJECT PLANNING**

Briefly, some common pitfalls in project planning in organizations include:

***Preparing only documents.*** As discussed earlier, creating documents is *not* the same as project planning. (*Remember*: Documentation is done to *organize* thoughts and information about a project’s plans and to *allow* information from the plans to be used as a reference for project stakeholders.) Many organizations, however, treat the project planning process as nothing more than the preparation of documents. Sometimes, with little or no thought, a past plan is converted to the plan for a new project. This lack of thought causes the focus on the planning aspects of a project to be reduced. Each aspect of a plan must be well considered and thought out *before* a plan is documented. The bottom line is that a plan must be implementable.

**Best practice:** Shift the focus from treating planning as documentation to using documentation as a tool to organize a project.

***Inadequate time for planning.*** Often, when an organization is in a hurry to begin a project, the start of the project will be rushed and inadequate time will be allotted to planning. Planning, however, is a crucial activity. Failing to allow enough time to permit adequate planning causes the plans to be less likely to be effective. Execution is also more likely to require deviations from the plan and a greater frequency of midcourse corrections. An organization is well served to remember the wise counsel of Abraham Lincoln about felling a tree, which is quoted at the beginning of this chapter.

**Best practice:** Allow adequate time for planning activities — plan for planning.

***No training or the wrong training.*** Project planning training for computer science students by educational institutions, if done at all, is rare. Even more unlikely is training for these students in the art and science of project planning. Therefore, when promoting or recruiting a programmer to become a project manager, training in the art and science of planning will be needed. (*Note*: Most individuals learn to use Microsoft Project from their peers or from senior members in the organization. Few are formally trained in Microsoft Project — and many of those confuse scheduling and planning. Individuals who do receive training in Microsoft Project rarely receive training about the theory and practice of PERT/ CPM, the basis of actually using the tool. Not uncommon is seeing Microsoft Project plans with hanging nodes and no resource constraints. Needless to say, these schedules are obviously in error and not being used to their full value.)

**Best practice:** Provide formal training in PERT/CPM and other project management tools to individuals who are vested with the responsibility of planning.

***Skipping reviews.*** Two types of reviews are essential for quality control: a peer review and a managerial review. A peer review is conducted by a person (or persons) who has similar experience in a similar role. A *peer* review looks very closely at the details, whereas a *managerial* review looks at the “big picture.” Both of these reviews have a significant value. Cutting either one short (but more frequently the peer review) would be compromising. Because planning is the initial stage of a project, errors that are undetected in the planning review process will likely have costly consequences for the project.

**Best practice:** Conduct peer and managerial reviews.

***Lacking a PMO or having an ineffective PMO.*** Because a PMO is a cost center that needs costly senior and human resources, many organizations have a PMO in name only. In this scenario, the “non-PMO” is more project administration than project management. This type of PMO does not assist SPMs, but instead actually causes a greater expenditure of resources. By demanding all sorts of data and analyses, this type of PMO typically becomes a hindrance for SPMs. (*Remember*: The PMO should collect data from the SPMs to carry out analyses in the most nonintrusive manner possible. These analyses are then supplied to SPMs and senior management to facilitate the corrective actions needed to keep the project on course.)

**Best practice:** Establish a robust and effective PMO based on a well-defined and well-implemented process framework.

***Lacking a knowledge repository or having a poorly organized knowledge repository.***Many organizations fail to take the development and maintenance of a knowledge repository seriously. The knowledge repository becomes a “dumping ground” for records from completed projects. To have a proper knowledge repository, resources (hardware, software, and human resources) need to be dedicated to the vital activity of maintaining a knowledge repository. Not only does a well-structured knowledge repository assist in ensuring project success, but it also provides a springboard for taking an organization to the next higher level of increased effectiveness.

**Best practice:** Have a well-structured knowledge repository.

**BEST PRACTICES IN SOFTWARE PROJECT PLANNING**

In addition to the best practices already described, a few additional best practices include:

***Process-driven planning***. A process-driven planning approach facilitates uniformity among SPMs in the project planning community of an organization. By providing templates to ensure that no important aspect is forgotten or overlooked, process-driven planning also facilitates more comprehensive planning. Defining the process planning process and then subjecting it to continuous improvement will hone organizational project planning skills and progressively improve planning to a stage in which the variances between planned and actual achievements are narrowed down to a minimum.

**Best practice:** Have a process-driven planning approach.

***Balanced planning***. When planning each project, strike a balance between “what is needed” and “what is mandated.” Although having a PMP is a “bare minimum” planning requirement, a PMP will be inadequate for many projects. A better option (unless a project is very small) is to prepare a minimum of three plans: a PMP, a CMP, and a QAP. Include other pertinent aspects as needed in these three documents. Based on the complexity, duration, and the person-month effort required to execute the project, the preparation of more detailed (or additional) plans may be needed. For example, as additional human resources are employed, the complexity of management increases. Having three plans (a PMP, CMP, and QAP) may be adequate if the number of teams in a project is one (one team consists of six to ten people), but if the number of teams increases beyond one, the number and rigor of these plans must increase.

**Best practice:** Create a balanced set of plans based on the type of project, the effort required to execute the project, the expected duration, and the number of teams working on the project. Refer to the organizational norms for the recommended set of plan documents.

***Norms for planning.*** For the estimation component of planning to be realistic, norms, especially for software estimation, resource estimation, and other software engineering activities, must be made available to project planners. Obviously these norms should form a part of the organization’s knowledge repository. Derivation of norms based on studies and periodic adjustment, taking into consideration actual achievements, goes a long way in ensuring effective planning.

**Best practice:** Use organizational norms from the knowledge repository for the estimation component of planning.

***Variance analysis.*** Once a project is completed, an analysis of the variances from estimates to actual achievements needs to be carried out. A variance analysis includes comparing the original plan to the actual achievements, eliminating abnormal achievements with assignable causes that are specific to the project, drawing the correct inferences from the data, and updating the organizational norms. Subjecting a completed project to variance analysis and then adding updates to the knowledge repository ensure that the knowledge repository contains reliable and credible information. Although variance analysis is an important step in the postmortem process, in many organizations, conducting a variance analysis is more often an exception rather than the rule.

**Best practice:** Conduct a variance analysis during project postmortems and update the knowledge repository.