Peopleware: Productive Projects and Teams, Third Edition

by Tom DeMarco, Tim Lister

Released June 2013

Publisher(s): Addison-Wesley Professional

ISBN: 9780133440706

Explore a preview version of *Peopleware: Productive Projects and Teams, Third Edition* right now.

O’Reilly members get unlimited access to live online training experiences, plus books, videos, and digital content from 200+ publishers.

.............................................

DWYT 5th ed.

**from Chapter 2**

**CREATING AND MANAGING THE WORKPLAN** pg 63

Once a project manager has a general idea of the functionality and effort for the project, he or she creates a workplan, which is a dynamic schedule that records and keeps track of all the tasks that need to be accomplished over the course of the project. The workplan lists each task, along with important information about it, such as when it needs to be completed, the person assigned to do the work, and any deliverables that will result. The level of detail and the amount of information captured by the workplan depend on the needs of the project, and the detail usually increases as the project progresses.

The overall objectives for the system should be listed on the system request, and it is the project manager’s job to identify all the tasks that need to be accomplished to meet those objectives. Th is sounds like a daunting task. How can someone know everything that needs to be done to build a system that has never been built before?

One approach for identifying tasks is to get a list of tasks that has already been developed and to modify it. There are standard lists of tasks, or methodologies, that are available for use as a starting point. As we stated in Chapter 1, a methodology is a formalized approach to implementing a systems development process (i.e., it is a list of steps and deliverables). A project manager can take an existing methodology, select the steps and deliverables that apply to the current project, and add them to the workplan. If an existing methodology is not available within the organization, methodologies can be purchased from consultants or vendors, or books such as this textbook can serve as a guide. Because most organizations have a methodology they use for projects, using an existing methodology is the most popular way to

create a workplan. In our case, because we are using a Unified Process-based methodology, we can use the phases, workflows, and iterations as a starting point to create an evolutionary work breakdown structure and an iterative workplan.

**Evolutionary Work Breakdown Structures and Iterative Workplans7**

Because object-oriented systems approaches to systems analysis and design support incremental and iterative development, any project planning approach for object-oriented systems development also requires an incremental and iterative process. In the description of the enhanced Unified Process in Chapter 1, the development process was organized around iterations, phases, and workflows. In many ways, a workplan for an incremental and iterative development process is organized in a similar manner. For each iteration, there are different tasks executed on each workflow. This section describes an incremental and iterative process using evolutionary WBSs for project planning that can be used with object-oriented systems development.

Evolutionary WBSs allow the analyst to develop an iterative workplan. First, evolutionary WBSs are organized in a standard manner across all projects: by workflows, phases, and then the specific tasks that are accomplished during an individual iteration. Second, evolutionary WBSs are created in an incremental and iterative manner. This encourages a more realistic view of both cost and schedule estimation. Third, because the structure of an evolutionary WBS is not tied to any specific project, evolutionary WBSs enable the comparison of the current project to earlier projects. This supports learning from past successes and failures. In the case of the enhanced Unified Process, the workflows are the major points listed in the WBS. Next, each workflow is decomposed along the phases of the enhanced Unified Process. After that, each phase is decomposed along the tasks that are to be completed to create the deliverables associated with an individual iteration contained in each phase (see Figure 1-16). Th e template for the first two levels of an evolutionary WBS for the enhanced Unified Process would look like Figure 2-17.

As each iteration through the development process is completed, additional iterations and tasks are added to the WBS (i.e., the WBS evolves along with the evolving information system).8

|  |  |  |
| --- | --- | --- |
| I. Business Modeling | V. Implementation | IX. Project Management |
| a. Inception | a. Inception | a. Inception |
| b. Elaboration | b. Elaboration | b. Elaboration |
| c. Construction | c. Construction | c. Construction |
| d. Transition | d. Transition | d. Transition |
| e. Production | e. Production | e. Production |
|  |  |  |
| II. Requirements | VI. Test | X. Environment |
| a. Inception | a. Inception | a. Inception |
| b. Elaboration | b. Elaboration | b. Elaboration |
| c. Construction | c. Construction | c. Construction |
| d. Transition | d. Transition | d. Transition |
| e. Production | e. Production | e. Production |
|  |  |  |
| III. Analysis | VII. Deployment | XI. Operations and Support |
| a. Inception | a. Inception | a. Inception |
| b. Elaboration | b. Elaboration | b. Elaboration |
| c. Construction | c. Construction | c. Construction |
| d. Transition | d. Transition | d. Transition |
| e. Production | e. Production | e. Production |
|  |  |  |
| IV. Design | VIII. Configuration and Change Management | XII. Infrastructure Management |
| a. Inception | a. Inception | a. Inception |
| b. Elaboration | b. Elaboration | b. Elaboration |
| c. Construction | c. Construction | c. Construction |
| d. Transition | d. Transition | d. Transition |
| e. Production | e. Production | e. Production |

FIGURE 2-17 Evolutionary WBS Template for the Enhanced Unified Process

For example, typical activities for the inception phase of the project management workflow would include identifying the project, performing the feasibility analysis, selecting the project, and estimating the effort. The inception phase of the requirements workflow would include determining the requirements gathering and analysis techniques, identifying functional and nonfunctional requirements, interviewing stakeholders, developing a vision document, and developing use cases. Probably no tasks are associated with the inception phase of the operations and support workflow.

A sample evolutionary WBS for planning the inception phase of the enhanced Unified Process, based on Figures 1-16 and 2-17, is shown in Figure 2-18. Notice the last two tasks for the project management workflow are “create workplan for first iteration of the elaboration phase” and “assess the inception phase”; the last two things to do are to plan for the next iteration in the development of the evolving system and to assess the current iteration. As the project moves through later phases, each workflow has tasks added to its iterations. For example, the analysis workflow will have the creation of the functional, structural, and behavioral models during the elaboration phase. Finally, when an iteration includes a lot of complex tasks, traditional tools, such as Gantt charts and network diagrams, can be used to detail the workplan for that specific iteration.

**Managing Scope**

An analyst may assume that a project will be safe from scheduling problems because he or she

carefully estimated and planned the project up front. However, the most common reason for

schedule and cost overruns — scope creep — occurs after the project is under way. Scope creep

happens when new requirements are added to the project after the original project scope was

defined and frozen. It can happen for many reasons: Users might suddenly understand the potential of the new system and realize new functionality that would be useful; developers might discover interesting capabilities to which they become very attached; a senior manager might decide to let this system support a new strategy that was developed at a recent board meeting.

Fortunately, using an iterative and incremental development process allows the team to deal with changing requirements in an effective way. However, the more extensive the change becomes, the greater the impact on cost and schedule. The keys are to identify the requirements as well as possible in the beginning of the project and to apply analysis techniques effectively. For example, if needs are fuzzy at the project’s onset, a combination of intensive meetings with the users and prototyping would allow users to “experience” the requirements and better visualize how the system could support their needs.

Of course, some requirements may be missed no matter what precautions are taken. However, the project manager should allow only absolutely necessary requirements to be added after the project begins. Even at that point, members of the project team should carefully assess the ramifi cations of the addition and present the assessment to the users. Any change that is implemented should be carefully tracked so that an audit trail exists to measure the change’s impact.

Sometimes changes cannot be incorporated into the present system even though they truly would be beneficial. In this case, these additions should be recorded as future enhancements to the system. The project manager can offer to provide functionality in future releases of the system, thus getting around telling someone “no.”

A couple of useful agile techniques to manage the scope of the project while attempting to satisfy the client are daily scrum meetings and the product backlog used with Scrum. Essentially a daily scrum meeting is a very short, typically fift een minutes, meeting that keeps the development team up to date as to the current status of the evolving system. The content of the meeting typically only covers what has been accomplished since the previous meeting, what will be accomplished before the next meeting, and what obstacles could come up that could prevent progress from being made. Also, new requested features could be brought up. However, all proposed additional features are simply added to the product backlog that could be considered during the next iteration or timebox (sprint in Scrum’s nomenclature). The product backlog is essentially a prioritized list of the functional requirements that will be completed during the current iteration. In Scrum, only the client is allowed to modify the product backlog. In this manner, the development team always has a list of the current set of critical requirements. As long as the project is relatively small, this approach to scope management is very effective.

**Timeboxing**

Another approach to scope management is a technique called timeboxing. Up until now, we have described task-oriented projects. In other words, we have described projects that have a schedule driven by the tasks that need to be accomplished, so the greater number of tasks and requirements, the longer the project will take. Some companies have little patience for development projects that take a long time, and these companies take a time-oriented approach that places meeting a deadline above delivering functionality.

Think about the use of word processing software. For 80 percent of the time, only 20 percent of the features, such as the spelling checker, boldfacing, and cutting and pasting, are used. Other features, such as document merging and creating mailing labels, may be nice to have, but they are not a part of day-to-day needs. The same goes for other software applications; most users rely on only a small subset of their capabilities. Ironically, most developers agree that typically 75 percent of a system can be provided relatively quickly, with the remaining 25 percent of the functionality demanding most of the time.

To resolve this incongruency, the technique of timeboxing has become quite popular, especially when using RAD and agile methodologies. This technique sets a fixed deadline for a project and delivers the system by that deadline no matter what, even if functionality needs to be reduced. Timeboxing ensures that project teams don’t get hung up on the final finishing touches that can drag out indefinitely, and it satisfies the business by providing a product within a relatively short time frame.

Several steps are involved in implementing timeboxing on a project. First, set the date of delivery for the proposed goals. The deadline should not be impossible to meet, so it is best to let the project team determine a realistic due date. If you recall from Chapter 1, the Scrum agile methodology sets all of its timeboxes (sprint) to thirty working days. Next, build the core of the system to be delivered; you will find that timeboxing helps create a sense of urgency and helps keep the focus on the most important features. Because the schedule is absolutely fixed, functionality that cannot be completed needs to be postponed. It helps if the team prioritizes a list of features beforehand to keep track of what functionality the users absolutely need. Quality cannot be compromised, regardless of other constraints, so it is important that the time allocated to activities is not shortened unless the requirements are changed (e.g., don’t reduce the time allocated to testing without reducing features). At the end of the time period, a high-quality system is delivered, but it is likely that future iterations will be needed to make changes and enhancements. In that case, the timeboxing approach can be used once again.

**Refining Estimates**

The estimates that are produced during inception need to be refined as the project progresses. This does not mean that estimates were poorly done at the start of the project; rather, it is virtually impossible to develop an exact assessment of the project’s schedule at the beginning of the development process. A project manager should expect to be satisfied with broad ranges of estimates that become more and more specific as the project’s product becomes better defined.

During planning, when a system is first requested, the project sponsor and project manager attempt to predict how long the development process will take, how much it will cost, and what it will ultimately do when it is delivered (i.e., its functionality). However, the estimates are based on very little knowledge of the system. As the system moves into the elaboration, more information is gathered, the system concept is developed, and the estimates become even more accurate and precise. As the system moves closer to completion, the accuracy and precision increase, until it is delivered.

According to one of the leading experts in software development,9 a well-done project plan (prepared at the end of inception) has a 100 percent margin of error for project cost and a 25 percent margin of error for schedule time. In other words, if a carefully done project plan estimates that a project will cost $100,000 and take twenty weeks, the project will actually cost between $0 and $200,000 and take between fifteen and twenty-five weeks.

What happens if you overshoot an estimate (e.g., analysis ends up lasting two weeks longer than expected)? There are a number of ways to adjust future estimates. If the project team finishes a step ahead of schedule, most project managers shift the deadlines sooner by the same amount but do not adjust the promised completion date. The challenge, however, occurs when the project team is late in meeting a scheduled date. Three possible responses to missed schedule dates are presented in Figure 2-19. If, early in the project, an estimate proves to be too optimistic, planners should not expect to make up for lost time—very few projects end up doing this. Instead, they should change future estimates to include an increase similar to the one that was experienced. For example, if the first phase was completed 10 percent over schedule, planners should increase the rest of their estimates by 10 percent.

9 Barry W. Boehm et al., “Cost Models for Future Soft ware Life Cycle Processes: COCOMO 2.0,” in J. D. Arthur and S. M. Henry (eds.), Annals of Soft ware Engineering: Special Volume on Soft ware Process and Product Measurement (Amsterdam: J. C. Baltzer AG Science Publishers, 1995).

**Managing Risk**

One final facet of project management is risk management, the process of assessing and addressing the risks that are associated with developing a project. Many things can cause risks: weak personnel, scope creep, poor design, and overly optimistic estimates. The project team must be aware of potential risks so that problems can be avoided or controlled well ahead of time.

Typically, project teams create a risk assessment, or a document that tracks potential risks along with an evaluation of the likelihood of each risk and its potential impact on the project (Figure 2-20). A paragraph or two is also included to explain potential ways that the risk can be addressed. There are many options: The risk could be publicized, avoided, or even eliminated by dealing with its root cause. For example, imagine that a project team plans to use new technology but its members have identified a risk in the fact that its members do not have the right technical skills. They believe that tasks may take much longer to perform because of a high learning curve. One plan of attack could be to eliminate the root cause of the risk—the lack of technical experience by team members—by finding the time and resources needed to

provide proper training to the team.

Most project managers keep abreast of potential risks, even prioritizing them according to their magnitude and importance. Over time, the list of risks will change as some items are removed and others surface. Th e best project managers, however, work hard to keep risks from having an impact on the schedule and costs associated with the project.

RISK 1: The development of this system likely will be slowed considerably because project team members have not programmed in Java prior to this project.

Likelihood of risk: High probability of risk.

Potential impact on the project: This risk will probably increase the time to complete programming tasks by 50 percent.

Ways to address this risk: It is very important that time and resources are allocated to up-front training in Java for the programmers who are used for this project. Adequate training will reduce the initial learning curve for Java when programming begins. Additionally, outside Java expertise should be brought in for at least some part of the early programming tasks. This person should be used to provide experiential knowledge to the project team so that Java-related issues (of which novice Java programmers would be unaware) are overcome.

RISK 2:

FIGURE 2-20 Sample Risk Assessment

**STAFFING THE PROJECT**

Staffi ng the project includes determining how many people should be assigned to the project,

matching people’s skills with the needs of the project, motivating them to meet the project’s

objectives, and minimizing the confl ict that will occur over time. Th e deliverables for this part

of project management are a staffi ng plan, which describes the number and kinds of people

who will work on the project, the overall reporting structure, and the project charter, which

describes the project’s objectives and rules. However, before describing the development of a

staffi ng plan, how to motivate people, and how to handle confl ict, we describe a set of characteristics

of jelled teams.

**Characteristics of a Jelled Team10**

Th e idea of a jelled team has existed for a long time. Most (if not all) student groups are not

representative of the idea of a jelled team, and you may have never had the opportunity to

appreciate the eff ectiveness of a true team. In fact, DeMarco and Lister point out that teams

are not created; they are grown. Typically, in class projects, students are assigned or asked to

form a group, which makes the ability to grow a team very limited. However, growing development

teams is crucial in information systems development. Th e whole set of agile soft ware

development approaches hinges on growing jelled teams. Otherwise, agile development

approaches would totally fail.

According to DeMarco and Lister,11 “[a] jelled team is a group of people so strongly knit that the whole is greater than the sum of the parts. The production of such a team is greater than that of the same people working in unjelled form.” They go on to state that a jelled “team can become almost unstoppable, a juggernaut for success.” When is the last time that you worked with a group on a class project that could be described “a juggernaut for success”? Demarco and Lister identify five characteristics of a jelled team.

First, jelled teams have a very low turnover during a project. Typically, members of a jelled team feel a responsibility to the other team members. This responsibility is felt so intensely that for a member to leave the team, the member would feel that they were letting the team down and that they were breaking a bond of trust.

Second, jelled teams have a strong sense of identity. In many classes, when you are part of a group, the group chooses some cute name to identify the group and differentiate it from the other groups. However, in this case, it is not simply the choosing of a name. It is instead evolving every member into something that only exists within the team. This can be seen when members of the team tend to do non–work-related activities together, e.g., do lunch together as a team or form a basketball team composed of only members of the development team. Third, the strong sense of identity tends to lead the team into feeling a sense of eliteness. Th e members of a jelled development team almost have a swagger about the way they relate to nonteam employees. Good examples that come to mind that possess this sense of eliteness outside of the scope of information systems development teams are certain sports teams, U.S. Navy Seal teams, or big city police force SWAT teams. In all three examples, each team member is highly competent in his or her specialty area, and each other team member knows (not thinks) that he or she can depend on the team members performing his or her individual jobs with a very high-level of skill.

Fourth, during the development process, jelled teams feel that the team owns the information system being developed and not any one individual member. In many ways, you could almost say that jelled teams are a little communistic in nature. By this we mean that the individual contributions to the effort are not important to a true team. The only things that matter are the output of the team. However, this is not to imply that a member who does not deliver his or her fair share will not go unpunished. In a jelled team, any member who is not producing is actually breaking his or her bond of trust with the other team members (see the first characteristic). The final characteristic of a jelled team is that team members really enjoy (have fun) doing their work. The members actually like to go to work and be with their team members.

Much of this can be attributed to the level of challenge they receive. If the project is challenging and the members of the team are going to learn something from completing the project, the members of a jelled team will enjoy tackling the project.

When a team jells, they will avoid the five dysfunctions of a team defined by Lencioni.Lack of trust is the primary cause of a team becoming dysfunctional. Lencioni describes four other causes of a team becoming dysfunctional that can come from the lack of trust. First, dysfunctional teams fear conflict, whereas members of a jelled team never fear conflict.12 Going to a member of a jelled team and admitting that you do not know how to do something is no big deal. In fact, it provides a method for the team member to help out, which would increase the level of trust between the two members. Second, dysfunctional teams do not have a commitment to the team from the individual members. Instead, they tend to focus on their

individual performance instead of the team’s performance. This can even be to the detriment of the development team. Obviously, this is not an issue for jelled teams. Third, dysfunctional teams try to avoid accountability. With jelled teams, accountability is not an issue. Members of a jelled team feel a high level of responsibility to the other team members. No team member ever wants to let down the team. Furthermore, owing to the bond that holds jelled teams together, no member has any problem with holding other members accountable for their performance (or lack of performance). Fourth, dysfunctional teams do not pay attention to the team’s results. Again, in this case, the cause of this dysfunction is that the individual members only focus on their individual goals. From a team management perspective, the team leader should focus on getting the goals of the team aligned; a jelled team will attain the goals.

**PRACTICAL TIP** Avoiding Classic Planning Mistakes As Seattle University’s David Umphress has pointed out, watching most organizations develop systems is like watching reruns of Gilligan’s Island. At the beginning of each episode, someone comes up with a cockamamie scheme to get off the island, and it seems to work for a while, but something goes wrong and the castaways fi nd themselves right back where they started—stuck on the island. Similarly, most companies start new projects with grand ideas that seem to work, only to make a classic mistake and deliver the project behind schedule, over budget, or both.

Here we summarize four classic mistakes in the planning and project management aspects of the project and discuss how to avoid them:

1. Overly optimistic schedule: Wishful thinking can lead to an overly optimistic schedule that causes analysis and design to be cut short (missing key requirements) and puts intense pressure on the programmers, who produce poor code (full of bugs).

Solution: Don’t inflate time estimates; instead, explicitly schedule slack time at the end of each phase to account for the variability in estimates.

2. Failing to monitor the schedule: If the team does not regularly report progress, no one knows if the project is on schedule.

Solution: Require team members to report progress (or the lack of progress) honestly every week. There is no penalty for reporting a lack of progress, but there are immediate sanctions for a misleading report.

3. Failing to update the schedule: When a part of the schedule falls behind (e.g., information gathering

uses all the slack in item 1 plus 2 weeks), a project team often thinks it can make up the time later by

working faster. It can’t. This is an early warning that the entire schedule is too optimistic.

Solution: Immediately revise the schedule and inform the project sponsor of the new end date or use timeboxing to reduce functionality or move it into future versions.

4. Adding people to a late project: When a project misses a schedule, the temptation is to add more

people to speed it up. This makes the project take longer because it increases coordination problems

and requires staff to take time to explain what has already been done.

Solution: Revise the schedule, use timeboxing, throw away bug-filled code, and add people only to work

on an isolated part of the project.

Based upon Steve McConnell, Rapid Development (Redmond, WA: Microsoft Press, 1996), pp. 29–50.

7 Th is material in this section is based on Walker Royce, Soft ware Project Management: A Unifi ed Framework (Reading, MA: Addison-Wesley, 1998).

10 Th e material in the section is based on T. DeMarco and T. Lister, Peopleware: Productive Projects and Teams, 2nd Ed. (New York: Dorset House, 1999); P. Lencioni, Th e Five Dysfunctions of a Team: A Leadership Fable (San Francisco: Jossey-Bass, 2002).

11 T. DeMarco and T. Lister, Peopleware: Productive Projects and Teams, 2nd Ed., p. 123.

12 When confl ict occurs, it is necessary to address it in an eff ective manner. We discuss how to handle confl ict later in the chapter.