Iterative Process Planning

Like software development, project planning requires an iterative process. Like software, a plan is an intangible piece of intellectual property to which all the same concepts must be applied. Plans have an engineering stage, during which the plan is developed, and a production stage, when the plan is executed. Plans must evolve as the understanding evolves of the problem space and the solution space. Planning errors are just like product errors: The sooner in the life cycle they are resolved, the less impact they have on project success.

Comprehensive project plans are highly dependent on numerous parameters, any one of which can have a significant impact on the direction of a project. Nevertheless, generic planning advice is sought by every software project manager as a skeleton from which to begin. This chapter is not a plan, a cookbook for a plan, nor a recipe for a plan. It is simply a rough model of a few dimensions, perhaps a starting point for a plan.

10.1 WORK BREAKDOWN STRUCTURES

A good work breakdown structure and its synchronization with the process framework are critical factors in software project success. Although the concept and practice of using a WBS are well established, this topic is largely avoided in the published literature. This is primarily because the development of a work breakdown structure is dependent on the project management style, organizational culture, customer

Key Points
- Projects can underplan and they can overplan. Once again, balance is paramount in the level of planning detail and the buy-in among stakeholders.
- The work breakdown structure is the “architecture” of the project plan. It must encapsulate change and evolve with the appropriate level of detail throughout the life cycle.
- Cost and schedule budgets should be estimated using macroanalysis techniques (top-down project level) and microanalysis techniques (bottom-up task level) to achieve predictable results.

A WBS is simply a hierarchy of elements that decomposes the project plan into the discrete work tasks. A WBS provides the following information structure:

- A delineation of all significant work
- A clear task decomposition for assignment of responsibilities
- A framework for scheduling, budgeting, and expenditure tracking

Many parameters can drive the decomposition of work into discrete tasks: product subsystems, components, functions, organizational units, life-cycle phases, even geographies. Most systems have a first-level decomposition by subsystem. Subsystems are then decomposed into their components, one of which is typically the software. This section focuses on software WBS elements, whether the software is the whole project or simply one component of a larger system.

### 10.1.1 Conventional WBS Issues

Conventional work breakdown structures frequently suffer from three fundamental flaws.

1. They are prematurely structured around the product design.
2. They are prematurely decomposed, planned, and budgeted in either too much or too little detail.
3. They are project-specific, and cross-project comparisons are usually difficult or impossible.

Conventional work breakdown structures are prematurely structured around the product design. Figure 10-1 shows a typical conventional WBS that has been structured primarily around the subsystems of its product architecture, then further decomposed into the components of each subsystem. Once this structure is ingrained in the WBS and then allocated to responsible managers with budgets, schedules, and expected deliverables, a concrete planning foundation has been set that is difficult and expensive to change. A WBS is the architecture for the financial plan. Just as software architectures need to encapsulate components that are likely to change, so must planning architectures. To couple the plan tightly to the product structure may make sense if both are reasonably mature. However, a looser coupling is desirable if either the plan or the architecture is subject to change.
10.1 WORK BREAKDOWN STRUCTURES

FIGURE 10-1. Conventional work breakdown structure, following the product hierarchy

Management
System requirements and design
Subsystem 1
  Component 11
    Requirements
    Design
    Code
    Test
    Documentation
    . . . (similar structures for other components)
Component 1N
  Requirements
  Design
  Code
  Test
  Documentation
  . . . (similar structures for other components)
Subsystem M
  Component M1
    Requirements
    Design
    Code
    Test
    Documentation
    . . . (similar structures for other components)
Component MN
  Requirements
  Design
  Code
  Test
  Documentation
Integration and test
  Test planning
  Test procedure preparation
  Testing
  Test reports
Other support areas
  Configuration control
  Quality assurance
  System administration
Conventional work breakdown structures are prematurely decomposed, planned, and budgeted in either too little or too much detail. Large software projects tend to be overplanned, and small projects tend to be underplanned. The WBS shown in Figure 10-1 is overly simplistic for most large-scale systems, where six or more levels of WBS elements are commonplace. The management team plans out each element completely and creates a baseline budget and schedule for every task at the same level of detail. On the other hand, most small-scale or in-house developments elaborate their WBSs to a single level only, with no supporting detail. The management team plans and conducts the project with coarse tasking and cost and schedule accountability. Both approaches are out of balance. In general, a WBS elaborated to at least two or three levels makes sense. For large-scale systems, additional levels may be necessary in later phases of the life cycle. The basic problem with planning too much detail at the outset is that the detail does not evolve with the level of fidelity in the plan. For example, it is impossible to lay out accurately in month 1—when the plan is being baselined, and before the architecture and test scenarios have been engineered—details of the test activities that are scheduled 18 months later.

Conventional work breakdown structures are project-specific, and cross-project comparisons are usually difficult or impossible. Most organizations allow individual projects to define their own project-specific structure tailored to the project manager's style, the customer's demands, or other project-specific preferences. With no standard WBS structure, it is extremely difficult to compare plans, financial data, schedule data, organizational efficiencies, cost trends, productivity trends, or quality trends across multiple projects. Each project organizes the work differently and uses different units of measure. Some of the following simple questions, which are critical to any organizational process improvement program, cannot be answered by most project teams that use conventional work breakdown structures.

- What is the ratio of productive activities (requirements, design, implementation, assessment, deployment) to overhead activities (management, environment)?
- What is the percentage of effort expended in rework activities?
- What is the percentage of cost expended in software capital equipment (the environment expenditures)?
- What is the ratio of productive testing versus (unproductive) integration?
- What is the cost of release N (as a basis for planning release N +1)?

### 10.1.2 Evolutionary Work Breakdown Structures

An evolutionary WBS should organize the planning elements around the process framework rather than the product framework. This approach better accommodates
the expected changes in the evolving plan and allows the level of planning fidelity to evolve in a straightforward way. The basic recommendation for the WBS is to organize the hierarchy as follows:

- First-level WBS elements are the workflows (management, environment, requirements, design, implementation, assessment, and deployment). These elements are usually allocated to a single team (as discussed in Chapter 11) and constitute the anatomy of a project for the purposes of planning and comparison with other projects.

- Second-level elements are defined for each phase of the life cycle (inception, elaboration, construction, and transition). These elements allow the fidelity of the plan to evolve more naturally with the level of understanding of the requirements and architecture, and the risks therein.

- Third-level elements are defined for the focus of activities that produce the artifacts of each phase. These elements may be the lowest level in the hierarchy that collects the cost of a discrete artifact for a given phase, or they may be decomposed further into several lower level activities that, taken together, produce a single artifact.

A default WBS consistent with the process framework (phases, workflows, and artifacts) is shown in Figure 10-2. This recommended structure provides one example of how the elements of the process framework can be integrated into a plan. It provides a framework for estimating the costs and schedules of each element, allocating them across a project organization, and tracking expenditures.

The structure shown is intended to be merely a starting point. It needs to be tailored to the specifics of a project in many ways.

- Scale. Larger projects will have more levels and substructures.

- Organizational structure. Projects that include subcontractors or span multiple organizational entities may introduce constraints that necessitate different WBS allocations.

- Degree of custom development. Depending on the character of the project, there can be very different emphases in the requirements, design, and implementation workflows. A business process re-engineering project based primarily on existing components would have much more depth in the requirements element and a fairly shallow design and implementation element. A fully custom development of a one-of-a-kind technical application may require fairly deep design and implementation elements to manage the risks associated with the custom, first-generation components.
ITERATIVE PROCESS PLANNING

A Management
   AA Inception phase management
      AAA Business case development
      AAB Elaboration phase release specifications
      AAC Elaboration phase WBS baselining
      AAD Software development plan
      AAE Inception phase project control and status assessments
   AB Elaboration phase management
      ABA Construction phase release specifications
      ABB Construction phase WBS baselining
      ABC Elaboration phase project control and status assessments
   AC Construction phase management
      ACA Deployment phase planning
      ACB Deployment phase WBS baselining
      ACC Construction phase project control and status assessments
   AD Transition phase management
      ADA Next generation planning
      ADB Transition phase project control and status assessments

B Environment
   BA Inception phase environment specification
   BB Elaboration phase environment baselining
      BBA Development environment installation and administration
      BBB Development environment integration and custom toolsmithing
      BBC SCO database formulation
   BC Construction phase environment maintenance
      BCA Development environment installation and administration
      BCB SCO database maintenance
   BD Transition phase environment maintenance
      BDA Development environment maintenance and administration
      BDB SCO database maintenance
      BDC Maintenance environment packaging and transition

C Requirements
   CA Inception phase requirements development
      CAA Vision specification
      CAB Use case modeling
   CB Elaboration phase requirements baselining
      CBA Vision baselining
      CBB Use case model baselining
   CC Construction phase requirements maintenance
   CD Transition phase requirements maintenance
10.1 WORK BREAKDOWN STRUCTURES

FIGURE 10-2. Default work breakdown structure
• Business context. Contractual projects require much more elaborate management and assessment elements. Projects developing commercial products for delivery to a broad customer base may require much more elaborate substructures for the deployment element. An application deployed to a single site may have a trivial deployment element (such as an internally developed business application) or an elaborate one (such as transitioning from a mission-critical legacy system with parallel operation, to achieve zero downtime).

• Precedent experience. Very few projects start with a clean slate. Most of them are developed as new generations of a legacy system (with a mature WBS) or in the context of existing organizational standards (with preordained WBS expectations). It is important to accommodate these constraints to ensure that new projects exploit the existing experience base and benchmarks of project performance.

The WBS decomposes the character of the project and maps it to the life cycle, the budget, and the personnel. Reviewing a WBS provides insight into the important attributes, priorities, and structure of the project plan. In performing project assessments and software management audits over the past several years, I have found the WBS to be the most valuable source of objective information about the project plan. While the software development plan and the business case provide a context for review, the WBS and the relative budgets allocated among the elements provide the most meaningful indicators of the management approach, priorities, and concerns.

Another important attribute of a good WBS is that the planning fidelity inherent in each element is commensurate with the current life-cycle phase and project state. Figure 10-3 illustrates this idea. One of the primary reasons for organizing the default WBS the way I have is to allow for planning elements that range from planning packages (rough budgets that are maintained as an estimate for future elaboration rather than being decomposed into detail) through fully planned activity networks (with a well-defined budget and continuous assessment of actual versus planned expenditures).

10.2 PLANNING GUIDELINES

Software projects span a broad range of application domains. It is valuable but risky to make specific planning recommendations independent of project context. It is valuable because most people in management positions are looking for a starting point, a skeleton they can flesh out with project-specific details. They know that initial planning guidelines capture the expertise and experience of many other people. Such guidelines are therefore considered credible bases of estimates and instill some confidence in the stakeholders.
Project-independent planning advice is also risky. There is the risk that the guidelines may be adopted blindly without being adapted to specific project circumstances. Blind adherence to someone else's project-independent planning advice is a sure sign of an incompetent management team. There is also the risk of misinterpretation. The variability of project parameters, project business contexts, organizational cultures, and project processes makes it extremely easy to make mistakes that have significant potential impact. Within this book, I have tried to provide an adequate context so that such misinterpretations can be avoided. To temper the project-independent discussions, Appendix D presents a very detailed case study of a specific real-world project. The case study provides a good example of a project that is 90% consistent with the project-independent planning guidelines given here. It also provides examples and rationale for several minor deviations from these guidelines.

Two simple planning guidelines should be considered when a project plan is being initiated or assessed. The first guideline, detailed in Table 10-1, prescribes a default allocation of costs among the first-level WBS elements. The second guideline, detailed in Table 10-2, prescribes the allocation of effort and schedule across the life-cycle phases. Given an initial estimate of total project cost and these two tables, developing a staffing profile, an allocation of staff resources to teams, a top-level project
ITERATIVE PROCESS PLANNING

Schedule, and an initial WBS with task budgets and schedules is relatively straightforward. This sort of top-down plan development is a useful planning exercise that should result in a baseline for further elaboration.

What is the source of the data in Table 10-1 and Table 10-2? Unfortunately, it is not a data bank of well-documented case studies of numerous successful projects that followed a modern software process. These data came mostly from my own experience, including involvement with software cost estimation efforts over the past decade that spanned a broad range of software projects, organizations, processes, and technologies.

Table 10-1 provides default allocations for budgeted costs of each first-level WBS element. While these values are certain to vary across projects, this allocation provides a good benchmark for assessing the plan by understanding the rationale for deviations from these guidelines. An important point here is that this is cost allocation, not effort allocation. To avoid misinterpretation, two explanations are necessary.

1. The cost of different labor categories is inherent in these numbers. For example, the management, requirements, and design elements tend to use more personnel who are senior and more highly paid than the other elements use. If requirements and design together consume 25% of the budget

<table>
<thead>
<tr>
<th>Table 10-1. WBS budgeting defaults</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FIRST-LEVEL WBS ELEMENT</strong></td>
</tr>
<tr>
<td>Management</td>
</tr>
<tr>
<td>Environment</td>
</tr>
<tr>
<td>Requirements</td>
</tr>
<tr>
<td>Design</td>
</tr>
<tr>
<td>Implementation</td>
</tr>
<tr>
<td>Assessment</td>
</tr>
<tr>
<td>Deployment</td>
</tr>
<tr>
<td><strong>Total</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 10-2. Default distributions of effort and schedule by phase</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DOMAIN</strong></td>
</tr>
<tr>
<td>Effort</td>
</tr>
<tr>
<td>Schedule</td>
</tr>
</tbody>
</table>
The cost and schedule estimating process

(employing people with an average salary of $100/hour), this sum may represent half as many staff hours as the assessment element, which also accounts for 25% of the budget but employs personnel with an average salary of $50/hour.

2. The cost of hardware and software assets that support the process automation and development teams is also included in the environment element.

Table 10-2 provides guidelines for allocating effort and schedule across the life-cycle phases. Although these values can also vary widely, depending on the specific constraints of an application, they provide an average expectation across a spectrum of application domains. Achieving consistency using these specific values is not as important as understanding why your project may be different.

10.3 THE COST AND SCHEDULE ESTIMATING PROCESS

Project plans need to be derived from two perspectives. The first is a forward-looking, top-down approach. It starts with an understanding of the general requirements and constraints, derives a macro-level budget and schedule, then decomposes these elements into lower level budgets and intermediate milestones. From this perspective, the following planning sequence would occur:

1. The software project manager (and others) develops a characterization of the overall size, process, environment, people, and quality required for the project.
2. A macro-level estimate of the total effort and schedule is developed using a software cost estimation model.
3. The software project manager partitions the estimate for the effort into a top-level WBS using guidelines such as those in Table 10-1. The project manager also partitions the schedule into major milestone dates and partitions the effort into a staffing profile using guidelines such as those in Table 10-2. Now there is a project-level plan. These sorts of estimates tend to ignore many detailed project-specific parameters.
4. At this point, subproject managers are given the responsibility for decomposing each of the WBS elements into lower levels using their top-level allocation, staffing profile, and major milestone dates as constraints.

The second perspective is a backward-looking, bottom-up approach. You start with the end in mind, analyze the micro-level budgets and schedules, then sum all these elements into the higher level budgets and intermediate milestones. This approach
tends to define and populate the WBS from the lowest levels upward. From this perspective, the following planning sequence would occur:

1. The lowest level WBS elements are elaborated into detailed tasks, for which budgets and schedules are estimated by the responsible WBS element manager. These estimates tend to incorporate the project-specific parameters in an exaggerated way.

2. Estimates are combined and integrated into higher level budgets and milestones. The biases of individual estimators need to be homogenized so that there is a consistent basis of negotiation.

3. Comparisons are made with the top-down budgets and schedule milestones. Gross differences are assessed and adjustments are made in order to converge on agreement between the top-down and the bottom-up estimates.

Milestone scheduling or budget allocation through top-down estimating tends to exaggerate the project management biases and usually results in an overly optimistic plan. Bottom-up estimates usually exaggerate the performer biases and result in an overly pessimistic plan. Iteration is necessary, using the results of one approach to validate and refine the results of the other approach, thereby evolving the plan through multiple versions. This process instills ownership of the plan in all levels of management.

These two planning approaches should be used together, in balance, throughout the life cycle of the project. During the engineering stage, the top-down perspective will dominate because there is usually not enough depth of understanding nor stability in the detailed task sequences to perform credible bottom-up planning. During the production stage, there should be enough precedent experience and planning fidelity that the bottom-up planning perspective will dominate. By then, the top-down approach should be well tuned to the project-specific parameters, so it should be used more as a global assessment technique. Figure 10-4 illustrates this life-cycle planning balance.

10.4 THE ITERATION PLANNING PROCESS

So far, this discussion has dealt only with the application-independent aspects of budgeting and scheduling. Another dimension of planning is concerned with defining the actual sequence of intermediate results. Planning the content and schedule of the major milestones and their intermediate iterations is probably the most tangible form of the overall risk management plan. An evolutionary build plan is important because there are always adjustments in build content and schedule as early conjecture evolves into well-understood project circumstances.
A generic build progression and general guidelines on the number of iterations in each phase are described next. Iteration is used to mean a complete synchronization across the project, with a well-orchestrated global assessment of the entire project baseline. Other micro-iterations, such as monthly, weekly, or daily builds, are performed en route to these project-level synchronization points.

- Inception iterations. The early prototyping activities integrate the foundation components of a candidate architecture and provide an executable...
framework for elaborating the critical use cases of the system. This framework includes existing components, commercial components, and custom prototypes sufficient to demonstrate a candidate architecture and sufficient requirements understanding to establish a credible business case, vision, and software development plan. Large-scale, custom developments may require two iterations to achieve an acceptable prototype, but most projects should be able to get by with only one.

- Elaboration iterations. These iterations result in an architecture, including a complete framework and infrastructure for execution. Upon completion of the architecture iteration, a few critical use cases should be demonstrable: (1) initializing the architecture, (2) injecting a scenario to drive the worst-case data processing flow through the system (for example, the peak transaction throughput or peak load scenario), and (3) injecting a scenario to drive the worst-case control flow through the system (for example, orchestrating the fault-tolerance use cases). Most projects should plan on two iterations to achieve an acceptable architecture baseline. Unprecedented architectures may require additional iterations, whereas projects built on a well-established architecture framework can probably get by with a single iteration.

- Construction iterations. Most projects require at least two major construction iterations: an alpha release and a beta release. An alpha release would include executable capability for all the critical use cases. It usually represents only about 70% of the total product breadth and performs at quality levels (performance and reliability) below those expected in the final product. A beta release typically provides 95% of the total product capability breadth and achieves some of the important quality attributes. Typically, however, a few more features need to be completed, and improvements in robustness and performance are necessary for the final product release to be acceptable. Although most projects need at least two construction iterations, there are many reasons to add one or two more in order to manage risks or optimize resource expenditures.

- Transition iterations. Most projects use a single iteration to transition a beta release into the final product. Again, numerous informal, small-scale iterations may be necessary to resolve all the defects, incorporate beta feedback, and incorporate performance improvements. However, because of the overhead associated with a full-scale transition to the user community, most projects learn to live with a single iteration between a beta release and the final product release.
The general guideline is that most projects will use between four and nine iterations. The typical project would have the following six-iteration profile:

- One iteration in inception: an architecture prototype
- Two iterations in elaboration: architecture prototype and architecture baseline
- Two iterations in construction: alpha and beta releases
- One iteration in transition: product release

Highly precedented projects with a predefined architecture, or very small-scale projects, could get away with a single iteration in a combined inception and elaboration phase and could produce a product efficiently with the overhead of only four iterations. A very large or unprecedented project with many stakeholders may require an additional inception iteration and two additional iterations in construction, for a total of nine iterations. The resulting management overhead may be well worth the cost to ensure proper risk management and stakeholder synchronization.

**10.5 PRAGMATIC PLANNING**

Even though good planning is more dynamic in an iterative process, doing it accurately is far easier. While executing iteration N of any phase, the software project manager must be monitoring and controlling against a plan that was initiated in iteration N – 1 and must be planning iteration N + 1. The art of good project management is to make trade-offs in the current iteration plan and the next iteration plan based on objective results in the current iteration and previous iterations. This concept seems, and is, overwhelming in early phases or in projects that are pioneering iterative development. But if the planning pump is primed successfully, the process becomes surprisingly easy as the project progresses into the phases in which high-fidelity planning is necessary for success.

Aside from bad architectures and misunderstood requirements, inadequate planning (and subsequent bad management) is one of the most common reasons for project failures. Conversely, the success of every successful project can be attributed in part to good planning. This book emphasizes the importance of three perspectives: planning, requirements, and architecture. The end products associated with these perspectives (a software development plan, requirements specifications, and an architecture description document) are not emphasized. On most successful projects, they are not very important once they have been produced. They are rarely used by most performers on a day-to-day basis, they are not very interesting to the end user, and their paper representations are just the tip of the iceberg with respect to the working details that underlie them.
While a planning document is not very useful as an end item, the act of planning is extremely important to project success. It provides a framework and forcing functions for making decisions, ensures buy-in on the part of stakeholders and performers, and transforms subjective, generic process frameworks into objective processes. A project's plan is a definition of how the project requirements will be transformed into a product within the business constraints. It must be realistic, it must be current, it must be a team product, it must be understood by the stakeholders, and it must be used.

Plans are not just for managers. The more open and visible the planning process and results, the more ownership there is among the team members who need to execute it. Bad, closely held plans cause attrition. Good, open plans can shape cultures and encourage teamwork.