Planning an Iterative Project

Philippe Kruchten

October 15, 2002

Rational Unified Process® expert Kruchten explains the combination of top-down and bottom-up planning required for iterative projects.

Planning an iterative project is both harder and easier than planning a waterfall project:

- It is harder, and much more work, because the planning is more dynamic and ongoing.
- It is easier, because an iterative approach is much more in tune with the way a project progresses. There is a short-term planning horizon, and plenty of opportunities to adjust the plans.

Traditional planning of engineering projects is far too often organized from the top down and around a product breakdown -- that is, the decomposition of the system into components and various artifacts (specifications, blueprints, subassemblies, etc.). This style of planning was inherited from the manufacturing and construction industries. Although at some point the plan needs to acknowledge the artifacts and the product structure, this is often done too early in the software development process, when too little is known about the product.

In the Rational Unified Process® (RUP®), planning is more focused on a process breakdown -- that is, what needs to be done to achieve certain objectives over time. You plan based on phases and iterations that include major and minor milestones. You use activities and workflow details. And you use both a top-down approach and a bottom-up approach: top down primarily in the early phases (Inception and Elaboration), complemented by bottom-up in later phases (Construction and Transition). The bottom-up approach uses the defined artifacts and the architectural baseline. In practice, the planning approach is not black and white; there is a top-down/bottom-up continuum across the development lifecycle.

This dynamic and iterative approach to planning allows you to achieve a better balance between target -- where you want to be at the end of the project -- and commitments -- what the team must do to get there. It is adaptive, and it is driven by risks and what has been learned so far, about both the product and the process used to develop it.

Note: if you are unfamiliar with the basics of iterative development, take a minute to review the "Key Concepts" presented below in the sidebar.
Key Concepts

In order to understand the details of planning an iterative development project, it is imperative that you become familiar with the basic concepts listed here.

- **Cycles** -- A development cycle is the period of time that elapses from the very start of a project until product release (or project cancellation); it includes all the activities executed during that time.

- **Phases** -- Each cycle in the RUP breaks down into a sequence of four phases: Inception, Elaboration, Construction, and Transition. Remember that these phases always exist, and that they are important not because of what is executed or because of their length, but because of what is achieved at the major milestone that concludes each one.

- **Iterations** -- Each phase may encompass one or more iterations. Software is developed in each iteration, and each concludes with a minor milestone including a release (internal or external) that is a point for assessing project progress. The software product grows incrementally as you iterate over the activities.

- **Builds** -- Within each iteration, the various developers or teams might produce software builds (sometimes at a high rate: daily or even more frequently), which allow for continual integration, testing, and regression testing, and also provide general indication of progress. Often the last build of the iteration is part of that iteration's release.

- **Time Boxing** -- Time boxing is a top-down technique for confining the achievement of a specific goal to a short timeframe. This provides a structure for focusing on the most important mission goals and forcing engineering trade-offs. It is not about setting impossible or irrational goals and delivering anything of poor quality within the time box. Instead, it is a tool used to balance scope and schedule, but also to force convergence, and limit analysis-paralysis or gold plating.

Not all cycles have the same development profile. The importance of the phases, and the number of iterations required to fulfill the phase objectives, can vary greatly. We often distinguish initial development cycles from evolution cycles or maintenance cycles that lead to product upgrades. An initial development cycle that leads to the very first release of a system is often referred to as a "green-field" development. Figure 1 shows a typical timeline for an initial development cycle.

![Figure 1: Typical Timeline for an Initial Development Cycle](image)

Coarse-Grained and Fine-Grained Plans: Project Plans and Iteration Plans

Because the iterative process is highly dynamic and adaptive, and is meant to accommodate changes to goals and tactics, there is no purpose in spending an inordinate amount of time producing detailed plans that extend beyond the current project horizon. Such plans are difficult to maintain, they rapidly become obsolete, and they are typically ignored by the performing organization after a few weeks. They try to be very predictive, and may use "inchpebbles" rather than milestones. Moreover, software development is a creative activity, not just a production activity or an administrative process with a more or less fixed workflow. It is hard to decide on a plan before you actually know what it is that you are planning.

In an iterative process, two kinds of plans are useful:
• A "coarse-grained" **Project Plan** that focuses on phases and iterations and their objectives, and on the overall staffing level.
• A series of "fine-grained" **Iteration Plans**, one per iteration, that bring RUP activities and individual resources into perspective.

### The Project Plan

Top managers and external stakeholders are rarely interested in the details of who is doing what and when. They mostly care about the release date, any milestones along the way, when major decisions must be made, and where they can get visibility into the progress, scope, difficulties, and resources of the project.

The **Project Plan** is a coarse-grained plan, and there is only one per development project. It captures the overall "envelope" of the project, for one cycle (and maybe the following cycles, if appropriate). The Project Plan includes the following:

• **Dates for the major milestones:**
  o **Lifecycle Objective (LCO) milestone** at the end of Inception, when the project is well scoped and funded.
  o **Lifecycle Architecture (LCA) milestone** at the end of Elaboration, when the architecture is complete and the requirements baseline is set.
  o **Initial Operational Capability (IOC) milestone** at the end of Construction, which marks the first "beta" release.
  o **Product Release (PR) milestone** at the end of Transition and the development cycle.
• **Staffing profile:** What resources are required over time?

### The Iteration Plan

An **Iteration Plan** is a fine-grained, time-boxed plan; there is one per iteration. As each Iteration Plan focuses on only one iteration, it has a time span small enough to let the planners do a good job of detailing tasks with the right level of granularity and allocating them to appropriate team members.

A project usually has two Iteration Plans "active" at any time:

• A plan for the **current iteration**, to track progress for the iteration that is underway.
• A plan for the **next iteration**, which is built toward the second half of the current iteration and is ready at the end of it.
While his team is executing the current Iteration Plan, the project manager is writing the plan for the next iteration. He needs to be "light on his feet" and able to quickly modify this second plan late in the current cycle so that the team can have a viable plan for the subsequent iteration, even if changes are brought about by late-breaking events. In such cases he needs to react quickly, so that the team is not left at the starting gate because of a planning gap.

The Iteration Plan is built using traditional planning techniques and tools (Gantt charts, etc.) to define the tasks and allocate them to individuals and teams. The plan contains some important dates, such as major builds, arrival of components from other organizations, and major reviews. It defines objective success criteria for the iteration. It consists of RUP activities, or sometimes aggregates of activities, such as workflow details. Because these activities are uniquely associated with a role, and certain individuals on the team might be competent to play this or that role, the plan will relate directly to these resources.

You can think of an Iteration Plan as a window into the Project Plan (or Phase plan), that also acts as a magnifier, as shown in Figure 2 below.

Figure 2: Phase (or Project) Plan and Iteration Plan

The Iteration Plan is a separate artifact in the RUP and is not maintained beyond the duration of the iteration.

Building a Project Plan

To build a Project Plan, you will need an initial estimate of the overall size of the project. (See the Appendix for this article.) Once you have that, there are well-known recipes (such as the COCOMO model) to estimate the right level of staffing.
Most projects will not function efficiently if they are staffed with a constant number of people. The staffing profile usually climbs from Inception to Construction, plateaus for a while, and then drops a bit toward the end of Transition.

**Figure 3: Typical Resource Profile for a Development Cycle**

Like many other aspects of the RUP, this profile should be taken with a grain of salt; you should adjust it to suit your own experience and context. For example:

- If you have many unknowns and risks, and/or new people, tools, and technologies, then the Elaboration phase might last longer.
- For evolution cycles, Inception and Elaboration might be shorter.

Once you have planted dates for your major milestones on the calendar, the next question is how many iterations you will need and how long each must be. Opinion varies greatly among iterative development practitioners. First, remember to not confuse a build, such as a weekly build, with an iteration. Many hard-core fans of iterative development think of an iteration as lasting two to five weeks. This might be fine for most small- to medium-sized projects, but not for large and very large projects that involve coordinating the work of hundreds of people who are often distributed across several sites.

How quickly you can iterate depends mostly on the size of the development organization. Here are some example scenarios.

- **Five people, one week.** A team of five people can do some planning on Monday morning, have lunch together every day to monitor progress and reallocate tasks, start doing a build on Thursday, and complete the iteration by Friday evening. The iteration takes one week.
- **Twenty people, three to four weeks.** The one-week scenario would be very hard to achieve with twenty people. It takes more time to distribute the work, synchronize between subgroups, integrate code, and so on. An iteration might take three to four weeks.
- **Forty people, eight weeks.** With forty people, it takes a week just to get all the synapses firing from the development organization's brain to its extremities. With intermediate levels of management, you will require more formal documentation and more ceremony to achieve a common understanding of the objectives. Two months is a more reasonable iteration length.

Other factors come into play: the organization's degree of familiarity with the iterative approach, whether it’s a stable and mature organization, and the level of automation the team is using to manage code (e.g., distributed CM), distribute information (e.g., an intranet), automate testing, and so on. Rational's development organization produces synchronized releases of our Suite products.
with a team of 700 people and iterations that are five to eight weeks long. Craig Larman of Valtech reports that his projects have ten iterations or more and take only a bit more than a year.

Also, be aware that there is some fixed overhead time attached to an iteration: for planning, synchronizing, analyzing the results, and so on.

Consequently, even if you are convinced of the tremendous benefits of the iterative approach and are tempted to iterate furiously, these human activities will slow you down. Also keep in mind that:

- For iterations of more than six months, you will probably need to build in intermediate milestones to keep the project on track. Consider reducing the scope of the iteration to reduce its length and ensure a clear focus.
- Iterations of more than twelve months in multiyear projects create additional business risks, because each iteration exceeds the annual funding cycle. A project that has not produced anything visible in the past twelve months is at risk of losing its funding. With such a timetable you would not reap much benefit from iterative development. Rethink your strategy instead.
- Iterations of less than one month need to be scoped carefully. Typically, short iterations are more suitable for the Construction phase, in which the degree of new functionality and novelty are low. Short iterations might involve little or no formal analysis or design and represent only incremental improvements to existing, well-understood functionality.

As the last point above implies, iterations need not all be the same length; their length will vary according to their objectives. Typically, Elaboration iterations will be longer than Construction iterations. But within a phase, iterations should generally be the same length (to make planning easier). It can also be helpful to establish a regular "tempo" for a project that has many iterations by making the iterations (sensibly) the same length across the lifecycle.

**Determining the Number of Iterations**

Closely related to the length issue is the delicate issue of the number of iterations.

A very simple project might have only one iteration per phase:

- Inception: **One iteration** to produce a proof-of-concept prototype or user-interface mock-up (or perhaps **no iteration** at all if the project is an evolution cycle).
- Elaboration: **One iteration** to produce an architectural prototype.
- Construction: **One iteration** to build the product (up to a beta release).
- Transition: **One iteration** to finish the product (full product release).

For a more substantial project, in its initial development cycle, the norm would be:

- Inception: **One iteration**, possibly to produce a prototype.
- Elaboration: **Two iterations**: the first to develop an architectural prototype and the second to finalize the architectural baseline.
- Construction: **Two iterations**: the first to expose a partial system and the second to mature it for beta test.
- Transition: **One iteration** to go from beta to full product release.
For a **large project, with many unknowns, new technologies**, and the like, there may be a case for additional iterations:

- **Inception:** An *additional iteration* to allow for more prototyping.
- **Elaboration:** An *additional iteration* to allow different technologies to be explored, or to phase in the introduction of architectural solutions.
- **Construction:** An *additional iteration* because of the sheer size of the product.
- **Transition:** An *additional iteration* to allow for operational feedback.

So over a development cycle, we have several possible degrees of iteration (see Table 1).

<table>
<thead>
<tr>
<th>Cycle Iteration Count</th>
<th>Total Iterations in Cycle</th>
<th>Number of Iterations per Phase [I, E, C, T]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>3</td>
<td>[0, 1, 1, 1]†</td>
</tr>
<tr>
<td>Typical</td>
<td>6</td>
<td>[1, 2, 2, 1]</td>
</tr>
<tr>
<td>High</td>
<td>9</td>
<td>[1, 3, 3, 2]</td>
</tr>
<tr>
<td>Very high</td>
<td>10</td>
<td>[2, 3, 3, 2]</td>
</tr>
</tbody>
</table>

* [0, 1, 1, 1] denotes: Inception 0 iterations; Elaboration 1 iteration; Construction 1 iteration; Transition 1 iteration. Note that “0” does not mean that no work is being performed in that phase (Inception, for example), but rather that no software is being built.

So, in general, plan to have between three and ten iterations. Observe, though, that the upper and lower bounds connote unusual circumstances; *most development cycles require six to eight iterations.*

Many variations are possible, depending on project risks, size, and complexity:

- **If the product is intended for a totally new domain,** you might need to add some iterations in Inception to consolidate the concepts, show various mock-ups to a cross-section of customers or end users, or build a solid response to a request for proposal.
- **If a new architecture must be developed, a large amount of use-case modeling must be done, or there are very challenging risks,** you should plan to have two to three iterations in Elaboration.
- **If the product is large and complex,** and to be developed over a long period, you should plan to have three or more iterations in Construction.
- **You should plan to have several iterations in Transition if you must minimize time to market and therefore deliver the product with a reduced set of functionality -- or if you feel you might need to make a lot of small adaptations because the end-user base will expand or change after a period of use.**
- **A simple evolution cycle can be done with few iterations:** none in Inception and Elaboration, one in Construction, and one in Transition.
- **An organization that has never done iterative development** may choose to start with a low number, such as three iterations.
Alternatively, you might settle on an iteration length that your organization is comfortable with, and just proceed by fitting as many iterations of that length as you can in each phase. This is easier in some ways, but you need to have done it before, either with your current organization or one with a comparable size and level of experience, to be certain about iteration duration. Again, there is no "one-size-fits-all" approach. Do not try to optimize too much. Take a first shot and then modify as you learn.

**Iteration Length**

As a first approximation, obtain the iteration length by dividing the length of the phase by the number of iterations. If the duration you obtain is not quite right according to the rules of thumb given above, revisit the process. Now is the time to make adjustments, either to the length of a phase, or to the number of iterations. Then also take into account major holidays, planned vacations, or plant closures to adjust the project plan to reality. For example, it is not a good idea to plan a major milestone between Christmas and New Year's Eve, in countries like the US or most European countries.

Remember, it’s not necessary get your plan 100 percent right on day one, because you will revisit this plan several times during the development cycle. Start with something realistic to create a baseline Project Plan, and revisit it as you grow wiser at the end of each iteration.

**Iteration Objectives**

Once you have an idea of the number of iterations in your coarse-grained Project Plan, you need to define the contents of each iteration. It is even a good idea to find a name or title for the release at the end of each iteration, which will help people get a better focus on the next target. For example, the iterations for developing a private telephone switch might be:

- Iteration 1: Local Station-to-Station Call
- Iteration 2: Add External Calls and Subscriber Management
- Iteration 3: Add Voice Mail and Conference Calls

Again, do not put too much effort into this. You will revise these names several times throughout the cycle. Planning is iterative and adaptive.

**Staffing the Project**

The next step is to allocate the right level of resources for each lifecycle phase. A typical staffing profile looks like the one in Figure 4.
Figure 4: Example Resource Profile Across Project Lifecycle Phases

Here again, your actual staffing profile will vary, depending on your project.

- For *an initial (green-field) development cycle*, it is not a good idea to start with too many people during Inception and Elaboration. With nothing in place, you will struggle to keep everyone fully employed. It doesn't take fifty people to shape the Vision Document, so keep staffing down in the early phases and let it peak during Construction.

- In an *evolution cycle*, the staffing profile might be more flat. The same five people might be permanently allocated to it, and the activities actually performed will be similar to Transition, or in a greenfield project, the activities of Construction and Transition. You will not see a large change in staffing level.

For each phase, models such as COCOMO will help you determine the optimal number of people versus phase duration, taking into account various parameters (cost drivers).

You will need the right mix of competencies to allocate activities to individual resources. Start by setting up a table that lists all team members; then, for each one, record which RUP roles he or she is capable of taking on.

**Optimizing the Project Plan**

Beyond the basic recipe for planning an iterative project, the bold project manager can attempt to optimize the project plan in a couple of ways.

**Overlapping Iterations.** A certain amount of overlap can occur between one iteration and the next, and this is probably healthy if it keeps everyone busy. Although planning for the next iteration should not wait until you come to a complete stop, you should also not lose sight of the fact that to reap the benefits from iterative development, you need the lessons learned from iteration N to do a better job in iteration N+1. Too much overlap will defeat this built-in mechanism for refining requirements, goals, and process.

**Parallel Iterations.** When a product either has many parts or is developed by a distributed team, it might be tempting to have each team or subcontractor do their own planning. This is fine, provided that the work packages are fairly independent. If they are not, then it is better to have everyone
work according to the same global clock, and to integrate the various subsystems together at the end of an iteration. In some cases, a team might not deliver anything new at the end of an iteration; what they delivered for the previous iteration might be sufficient for the other group(s) to proceed. For example, a group responsible for infrastructure code might not deliver anything after an iteration if the infrastructure is already rather mature and its interface is stable, and if the developers do not release a new infrastructure with each iteration. If you do parallel iterations, remember that the slowest team or group is the limiting factor: It can slow down everyone else and force the other teams to synchronize with its schedule.

**Building an Iteration Plan**

The development of an *Iteration Plan* has four steps:

1. **Determine the iteration scope.** Determine the intent -- i.e., what you want to accomplish in the iteration.
2. **Define iteration evaluation criteria.** Define how to objectively evaluate the accomplishments at the end of the iteration; specify which artifacts will be worked on.
3. **Define iteration activities.** Establish what work needs to be done, and on which artifacts.
4. **Assign responsibilities.** Allocate resources to execute the activities.

The best way to proceed with the first three steps will vary greatly across the lifecycle.

**Inception and Elaboration**

At the beginning of a project, especially a green-field project, you will not yet have identified design elements specific to the new system on which to base your planning the iteration. So instead, use a top-down approach, with rough estimates derived from other projects as a basis for your planning assumptions; in fact, you probably did just this for the Project Plan.

If, however, you are dealing with an evolution cycle for an existing software product (maybe we should call this brown-field), Inception and Elaboration are likely to be shorter, you may have fewer risks to mitigate, and planning iterations will be more like those described below under *Construction and Transition*.

The objectives of the iteration will be determined primarily by risks, which will, in turn, determine which use cases, scenarios, algorithms, and so on, will be developed during the iteration. The risks will also determine what means to use in order to assess whether the risks have been mitigated.

**Construction and Transition**

By the time you have an overall architecture in place, some measurements from previous iterations relating to artifacts (lines of code, defects, etc.) and process (time to complete certain tasks), hopefully, you will also have mitigated most of your risks. You can use this information to progressively modify the way you plan iterations.

Now, you can proceed with a bottom-up, artifact-based planning approach, using design elements such as class, component, subsystem, use case, test case, and so on, as well as any measures
from previous iterations to estimate the effort. A word of warning: A bottom-up approach tends to be pessimistic relative to the schedule, especially when summing up the individual estimates of dozens of people. And it is known that novice developers have a tendency to be overly optimistic about their performance.

The iteration objectives will be determined primarily by completion objectives, and achievement of a set level of quality. This also includes specific defects to correct, primarily those that prevent the use of major functionality or make the system crash; it also includes deferring "nice to haves" for future releases.

**Identifying Activities**

Use your RUP Development Case as the reference list for activities required within an iteration. If you find the activities too granular, you may replace them with workflow details, especially in the early stages.

There are some activities that need to be run only once per iteration (or per phase or even per cycle). Both *Plan an Iteration* and *Lifecycle Milestone Review* fall into this category.

But other activities must be instantiated (replicated) for each element, and the element is usually the activity's major output. For example: *Code a Class* must be done for each class; *Integrate Subsystem* must be done for each subsystem and *Describe Use Case* must be done for each use case.

Consequently, in the very early iterations of a new project, because the design elements have not been identified, you will only be able to assign a "ballpark" figure to each activity. For example:

- Code (all) classes: 10 person-days

Or, for a higher-level artifact:

- Develop proof-of-concept prototype: 42 person-days

In later iterations, when the design elements are identified, activities can be associated to these elements with finer estimates. For example:

- Code *Customer* class: 2 person-days
- Code *Invoice* class: 1 person-day

**Plans Can Shape Cultures**

The Project Plan and Iterative Plans I've described are integral to the best practices inherent in the RUP; although good planning can be time consuming at the beginning of a project, the effort pays off in time saved later on. As a parting thought, I'll quote from Walker Royce:

Plans are not just for managers. The more open and visible the planning process, 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.
Acknowledgments
Thanks to Per Kroll, Joe Marasco, John Smith, and Walker Royce for their feedback on this article.

References


Also check the archives of *The Rational Edge* (www.therationaledge.com) for management articles, in particular the lively pieces by Joe Marasco on his experience managing software projects (Franklin's Kite section).

Notes

Appendix: Estimating the Project

An issue that affects both coarse-grained and fine-grained plans is the thorny issue of estimation. How big is this project? How many people are needed?

By far the best tool a project manager can use to estimate a project, phase, iteration, or simple activity is historical data. The most basic approach is to record effort, duration, and size estimates as well as your estimating processes and assumptions, and then track actual results against these estimates so you can generate more accurate estimates in the future. Estimating procedures and templates that itemize tasks can also help you avoid the common problem of overlooking necessary work. The RUP provides such tools.

Estimation tools such as COCOMO will not tell you how much effort is required for your new project; they will only help you estimate how long the project will take and how many people you will need when you adjust the numbers for various cost drivers, integrate the performance of your team, and assess the difficulty of the project. For the Project Plan, early in Inception, it is best to calibrate the new project relative to another one for which you know the total effort. Then a model like COCOMO will derive an estimation of duration and staffing level. Remember that software developers are notoriously nonfungible; this is what makes planning software projects a bit more tricky.

Once the project is started, you might want to combine top-down and bottom-up approaches, and exploit not only your growing knowledge of what you want to accomplish, but also, as the understanding of your development staff vis-à-vis the tools, their own performance, and the process, based on their experience with early prototyping. The section below describes a simple but effective approach for doing this. More sophisticated methods include using function points and various derivatives. Use cases can be used as a starting point for function-point-like techniques to estimate the "size of the mountain."

An Iterative Estimation Technique: Wideband Modified Delphi

Barry Boehm introduced this method, often referred to as the Wideband Modified Delphi, in the 1970s, and it has spawned several variants. The basic principle is to bring several heads together on an issue and try to reach a consensus. When the heads comprise the actual development team, the resulting estimates are more likely to get the team's buy-in than would apparently random numbers raining down from above. This is very roughly how it works:
1. The project manager defines what is to be estimated, the units of measure, and the assumptions. She gathers data for similar tasks or projects if available -- for example, data from previous iterations or a previous project. She selects participants.
2. All the participants are briefed on the procedure and goals, and given any available data.
3. The participants develop their own estimate (each one on his or her own), preferably not interacting with each other.
4. The project manager gathers all the data, tabulates it in a spreadsheet, and compares the numbers.
5. All participants meet again. If the numbers match, then fine; you have a likely estimate. If the numbers are widely scattered, it is interesting to discuss with participants the thinking behind them. When one team member explains his or her assumptions, others might point out some important parameter that was forgotten, some new risk that has arisen, and so on.
6. The participants are then given a chance to adjust their estimate based on the discussion.
7. The new numbers become the working estimate.

As the phase or iteration unrolls, actual data is collected for these tasks. When it is time to do another estimate(e.g., for the next iteration), participants consult the previous estimates and actuals to help them adjust for their natural optimism or pessimism.

There are plenty of variants and refinements, as you can imagine. You could iterate on steps 5 and 6 above, although it is often not necessary. I have done it very informally, using e-mail or simply walking from cubicle to cubicle with a notepad to discuss planning hypotheses with the people whose estimates varied widely. You can also take a more formal approach, using templates and tools to compute ranges and uncertainties, or even Monte Carlo simulations to generate a probability distribution of possible estimate outcomes, based on the final estimate values. A2
