Driving Iterative Development With Use Cases

Kurt Bittner

March 14, 2006
(First published March 25, 2004)

Deriving from a RUC 2003 presentation, this article explores how to plan a project based on the IBM Rational Unified Process (RUP) using a use-case based approach. Patterns for planning iteration in each phase are presented, providing you with a simple and practical approach to organizing the project work.

A Brief Overview of Iterative Software Development

Everyone struggles with how to find and write use cases, develop them, detail them, and so on. Once you have mastered these skills, the next questions are:

- When do I do it?
- In terms of the Rational Unified Process® (RUP®) phases -- Inception, Elaboration, Construction, and Transition -- when do things happen?
- Do I do work on use case A in iteration one, and use case B in iteration 2 (and so on)?

This article provides some background on iterative development in general, and then discusses how to drive the iterations with use cases and how to work on parts of use cases in different iterations. We'll also talk about what to do in the various RUP phases (such as which tasks you should complete in Inception, in Elaboration, and so forth).

Before we get to that, let's make sure that we understand RUP's phases, as well as the concept of how iterations are grouped into phases. RUP's phases, especially, tend to be misunderstood: people mistakenly equate Inception with Requirements, Elaboration with Analysis and Design, Construction with Implementation, and Transition with Deployment. The latter is almost true, but the other statements could not be more false. First, however, let's discuss planning the overall project lifecycle.

Planning the Overall Project

In a project following a "Waterfall" process lifecycle, the typical approach is as follows:

1. Capture all the requirements
2. Analyze those requirements
3. Define the design
4. Code the components of the solution
5. Integrate the components
6. Test the integrated solution.

There can be feedback loops between these steps, but the dominant practice is that a particular discipline is mostly completed before moving on to the next discipline. In contrast, using an iterative process you accomplish a small amount of each of the RUP disciplines (requirements, analysis, design, implementation, and testing) in every iteration. However, when you try to implement this process, you may run into immediate difficulties in the Inception Phase, as it is often unclear what (or how much) to do, when to do it, and how to drive the process with use cases. A better understanding of the RUP phases will mitigate this challenge.

One of the most positive aspects of the Iterative development process, on the other hand, is that it addresses the problem of not knowing where you really are until you start trying to integrate the parts (fairly far into the project), only to find out that things do not fit together quite as well as you thought they would. An iterative approach addresses certain kinds of risks sooner by implementing and integrating risky components earlier in the process. The reason this reduces risk is that you can really only know whether you have mitigated a risk after you have actually tried to build something and put the pieces together. Figure 2 illustrates some important differences in the two approaches.
The way that we confront risk using an iterative approach with use cases is first to identify our risks, and then use those risks to create use case scenarios that will force us to confront those risks. We should then analyze, design, implement, and test one or more of these scenarios in any given iteration. To avoid having to implement large amounts of the system too early, you might determine which parts of the system are architecturally insignificant or less time sensitive, and then stub\(^1\) in those parts in the early iterations. In this way, you avoid spending time building parts of the system that don't help you reduce risk. The net effect of this approach is that-by implementing the risky parts of the system first -- we have objective measures of progress in mitigating risks very early in the project.

The RUP divides the project into four phases, each of which is focused on a specific set of risks. Figure 3 presents a summary of the phases.

**The RUP Phases**

<table>
<thead>
<tr>
<th>RUP Phase</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td>Confirm the scope and objectives of the project and bring the business risks under control</td>
</tr>
<tr>
<td>Elaboration</td>
<td>Stabilize the product plans and bring the architectural and technical risks under control</td>
</tr>
<tr>
<td>Construction</td>
<td>Build the product and bring the logistical, project execution risks under control</td>
</tr>
<tr>
<td>Transition</td>
<td>Deliver the product and bring the roll-out risks under control</td>
</tr>
</tbody>
</table>

Activities in each phase focus primarily on reducing risks and moving the project ahead. Each phase conclusion represents a milestone, a point at which you should evaluate results and re-evaluate the decision to move ahead with the project. Figure 4 shows a summary of the milestones and their relationship to phases.
In the following sections, we will briefly describe the focus of -- and activities performed in each phase.

**The Inception Phase**

The purpose of the Inception phase is to bring business risks under control. What we mean by this is that we want to make sure that we:

- Understand the problem that we're solving
- Have confidence that we have an economically and technically viable approach for delivering a solution

To describe the problem's solution, we need a way to express what the system will do -- we typically do this by identifying (and briefly explaining, but not detailing) the use cases the system will have to support. Furthermore, to assess the technical viability of the system, we need to explore and quantify enough of the solution to understand whether it is possible, and how much it will cost (time and money). Only then can we discern whether the project is worth doing.

By the end of Inception we will be reasonably certain that the project is viable -- we know that:

- We can build the system
- It solves the intended problem
- We can do it within the target date and budget of the organization

We still don't know details about what we need to do or how we are going to do it, but we do know that our project is feasible.

**The Elaboration Phase**

The purpose of the Elaboration phase is to bring the architectural and technical risks under control by:

- Taking a deeper look at the solution we proposed in the Inception phase
- Examining whether it really is feasible

The basic approach we take is to build the complex, technically novel, or risky parts of the solution. We do this to make sure that we are not being overly optimistic in our belief that the solution will work. The emphasis here is on actually building and testing parts of the system, not just reviewing designs and plans. By the end of the Elaboration phase we will have had to confront issues such as:

- Performance
- Scalability
- Reliability
- Other technical issues

In this way we ensure that our proposed solution will actually work.
Even if we believe that we already have a proven architectural approach -- either because our solution has been used to solve similar problems, or because we are building the solution using middleware that has pre-defined architecture -- the Elaboration phase still plays an important role. In this situation we would want to evaluate whether the proposed (or candidate) architecture can mitigate the technical risks that we have identified. Until we can prove that the chosen approach is technically viable, we are still in the Elaboration phase (regardless of what the schedule says).

The Construction Phase
In Construction, we bring the logistical risks of the project under control. We have a large amount of work to get done, but if the business and technical risks are controlled, then it’s just a question of being able to do all the work:

- In the time allotted
- Within the budget
- With the resources allocated

You might need to do some scope management, but in general the focus of Construction should be cranking the work out. You can manage a large-scale project by using teams working in parallel, with integrated builds, branching, and merging changes using a change management system. At the end of Construction, you have a usable solution, but it is not yet fully deployed to its users.

The Transition Phase
The purpose of Transition is to roll the solution out into its target user community. In Transition you must:

- Train the users
- Establish back up and recovery policies
- Instantiate the solution at a data center
- Convert data, and so on

In other words, we need to take care of all the practical, essential activities that need to occur before a solution can be implemented.

The Distribution of Work Across Phases

As mentioned previously, a common misconception is that the Inception phase is mostly concerned with Requirements, the Elaboration Phase with Design, the Construction phase with
Implementation, and the Transition phase with Deployment. Figure 5 depicts the fact that each of the disciplines plays a role in each of the phases. For example, while we perform 40% or 50% of the requirements work in the Inception Phase, we also do some design and implementation, some deployment, and some management overhead (planning, estimating and measuring). The focus of the design and implementation work in Inception is on proof-of-concept prototypes that explore the business viability of proposed solutions.

In Elaboration (the next Engineering Phase), we conclude perhaps 60 -- 70% of the requirements work, half the design, and half the overall management effort. By Construction, requirements should be mostly complete -- you're getting ready to roll out the solution in the form of a beta release -- but there may be a few lingering issues. By the time you finish the Transition Phase, the work is complete.

Each phase concludes with a major milestone that corresponds to certain kinds of risk being mitigated. Each iteration concludes with a minor milestone (see Figure 6) at which we assess the results achieved in that iteration. The iteration assessment may result in us identifying new work, so we need to take this into account when creating a plan for subsequent iterations.

The phase and iteration milestones act as evaluation points we can use to make sure that we're on track, allowing the project to adjust to change and adapt to new information.
In each iteration, we take some set of scenarios and go through each of the RUP disciplines: Requirements, Analysis & Design, Implementation, and Test (illustrated in Figure 7). As an example, consider what deployment means in an Inception phase. We might do a proof-of-concept prototype, show it to some people, and ask if it is what they want. This doesn't mean that we are writing production-quality code in this iteration of the Inception phase, and it does not mean that we are doing exhaustive release testing, but it does mean that we do some implementation and testing in addition to all of the other work that we need to do. As we progress through the Elaboration phase and into the Construction phase, we are increasingly creating code that will survive in the final product.

How do use cases figure into this? Figure 8 shows a simple model of how use cases drive the behavior of the system. The relative size of the description shown in the box labeled "Use Cases" represents the fact that generating a description is most of the work involved in creating a use case. The little white box around some of the text in this document symbolizes a scenario. A scenario is a subset of a use case -- specifically, the basic flow plus zero or more alternate flows.
On the upper right we see a sequence diagram. This part of the figure is intended to show that each scenario has an associated collaboration that "fulfills" or "realizes" the scenario. Components participating in these collaborations interact with each other to realize or to provide the behavior specified by the use case. On the lower right we see that associated with each use case realization there are test cases that test either the scenario (white-box tests based on the use-case realization) or the use case itself (black-box tests based on the use-case description). Elsewhere in the figure we see that documentation and UI design, among other things, can be derived from use cases.

That's a brief overview of iterative development and its major concepts. It's important to understand these concepts in order to understand the details of how we can successfully complete the actual project work.

Planning Projects

Planning occurs at multiple levels. At the highest level is the Program Plan. A program coordinates multiple projects. To understand when a program would be useful, consider building a new kind of ATM that can dispense cash, provide theater tickets, and allow users to pay their parking tickets. If we were going to deploy this product across a nationwide (or perhaps worldwide) banking network, we would have to accomplish many high-level tasks:

- Coordinate work across many different systems and locales
- Change the software currently running in the financial institutions
- Coordinate that change with the rollout of the ATM software and hardware
- Coordinate all of the system changes with a marketing program that will educate consumers about the new product offering

The coordination of work across all the projects needed to do the work would require a program. To keep things simple for the rest of our discussion, we will focus on a single project.

The Project Plan is an achievement-oriented plan for the overall project that:

- Defines the dates of the major milestones (phase ends) of the project
• Maps risks to the phases in which they will be addressed
• Maps scenarios to the phases in which they will be addressed
• Provides a high-level overview of the resources needed in each phase (this will be refined in each of the phase plans)

The overall project plan delegates the details to the lower level plans -- phase and iteration plans. Often, it is very natural to combine the project plan and the phase plans.

In addition to the Project Plan, there is a detailed plan for each iteration describing the work that we will do in that iteration. An Iteration Plan identifies:

• The scenarios on which to focus
• The nonfunctional requirements to handle
• The test cases we will execute to validate the scenarios

To create the iteration plan, we consider the risks that are mapped onto the iteration, and then select scenarios and supplementary (typically non-functional) requirements that will force confrontation of those risks. From this starting point, we map out a series of activities to take these scenarios and supplementary requirements through the RUP disciplines discussed previously (requirements, analysis, design, implementation, test, and deployment).

Figure 9 shows the relationship between plans, moving from a high-level plan (of a program consisting of multiple projects) to project plans, to phase plans, and then to iteration plans.

**Planning, Estimation and Risk**

Most project management approaches consider risk and create a risk list. The difference between these conventional approaches and an iterative approach is the degree to which the risk list drives the overall project work. In the RUP, the risk list is the main driver of the planning effort, second only to the overall project objective. Aggressively confronting risk is the central theme of planning in the RUP. Risk reduction drives everything, including decisions about how we use (or even if we should use) use cases, as well as how much detail we should put in the use case descriptions.
Many people try to develop a detailed project plan at the beginning of the project, and then try to manage the project to this detailed plan. Using this approach, most projects end up over budget or late -- mostly not because the work was bad, but because the original plan and budget were bad. Figure 10 shows the variability in estimates over the course of the project. Most initial project plans are little more than conjectures, because there is little information on which to base estimates -- at the start of things, we don't know very much, and what you think you know may be wrong. As we get deeper into the project we get more information, which reduces the variability of the estimates. In addition, there is less work to estimate, and that further reduces errors.

We can use various approaches to develop estimates, and in fact different estimation models are useful at different points in the project. At the very beginning of the project (when there is little more than a notion that there is a problem and a sketchy vision for one or more solutions), all we may be able to do is to base estimates on hunches, which in turn are based on comparisons to similar projects. A little later in the Inception phase and into the early Elaboration phase (when we have a little more information about the requirements for the solution), techniques like function point analysis -- or its close cousin use-case point analysis -- can give you a better idea of the overall effort. By the time we are well into Elaboration, we have the emerging architecture to work with, and we can use more sophisticated estimation models such as CoCoMo II as the basis for our estimates. The Transition phase is driven by a completely different set of factors -- things like the number of sites and the number of users to which the solution will be deployed. As we go through each iteration, we periodically revise our estimates to ensure that our expectations are still realistic.

**A Pattern for Organizing Work in an Iteration**

As introduced previously in *Figure 7*, the general pattern of iteration is to more completely describe the requirements related to the risks assigned to the iteration, then analyze, design, implement, and test the part of the system needed to support the scenarios associated with the iteration’s risks. Figure 7 implies a rather strict sequence of activities, but the reality is far more flexible. In addition, a great deal more happens in parallel.

Many project management approaches specify a standard work breakdown structure, and in the pattern for work performed in an iteration described above we can see the basic building blocks for planning an iteration. In a simplified way, the work of an iteration can be viewed as shown in *Figure 11*. **Figure 10: Estimate accuracy increases as the project progresses**
Measuring Phase and Iteration results

At the end of each iteration, we need to take stock of what we have achieved:

- Have we mitigated the risks we set out to address?
- Have we implemented the parts of the solution needed to support the scenarios we selected for the iteration?
- If we didn't accomplish as much as we thought we would, how will we adjust our plans for future iterations?
- Can we still get all the work done, or do we need to consider reducing the project scope?

In evaluating results and adapting our plans for the future, we need to take care not to ignore signs that we are falling behind. When we don't succeed in mitigating a risk, it's easy to keep putting it off and saying that we will catch-up later -- but of course when we do this we take the first step toward serious trouble. When we don't achieve intended results in an iteration, we need to take a serious look at the scope of the project, and at the very least rethink our plans for future iterations.

Even more serious trouble starts when we don't achieve our targets for a phase, but keep moving ahead anyway. Phases are achievement-based -- you should never move on from one until you have mitigated the risks associated with that phase, regardless of what your schedule says. This is where we most often come into trouble -- by ignoring the warning signs, we set ourselves up for future failure. We will discuss this more later.

Planning and Managing the Inception Phase

Some people struggle with the purpose of the Inception phase. Our earlier statement that the Inception phase is focused on mitigating "business" risks seems confusing because the development team assumes that the business stakeholders are handling the business decisions and risks⁴. We have found that even when it appears that someone else is looking after the business issues, having the developers invest in learning about the business goals and risks results in the definition of better solutions. The common case is that there are a lot of issues that
fall right between business and technical issues -- availability, security, performance, throughput, and so on. In this case, an awareness of both sides of an issue leads to better results. The Inception phase provides a framework for collaboratively working through these issues from both the business and technical perspectives.

The overall project plan establishes the basic timetable for the phase, and identifies specific risks that will be handled in the phase. Since the Inception phase focuses on mitigating business risks, the Inception phase focuses on ensuring that the rest of the project, if funded, will result in something valuable to the organization. This usually means that the project will create a positive return on investment, satisfy a mandatory (perhaps legislated) requirement, or meet some time-to-market requirement. In the Inception phase, we identify problems and their root causes, and then develop a vision for their solution.

The work in the Inception phase tends to organize into tracks based on three kinds of activities:

- The Problem Track: Activities related to understanding the problem
- The Solution Track: Activities related to identifying potential solutions
- The Project Track: Activities related to organizing the project that will build the solution

These tracks work largely in parallel and interact with each other -- the problems to be solved affect the range of potential solutions, which in turn affects the way the project organized to deliver the solution is run. In turn, the way the project is organized and funded often constrains the solution that we can build, and so on.

The Problem Track

In the Problem track we try to get at the root causes of problems. For instance, the starting point of most projects is when symptoms of some problem present themselves -- costs are too high, customers are not satisfied, customer needs are unmet. We need to understand what the real problems are before we can propose solutions. A variety of techniques can be applied, most of which center around continually asking, "Why is that?" when we find a problem. When we get to the point where there are no more "whys", we are ready to find the right solution.

To understand the problem more fully, we usually need to know who experiences it. By considering who is affected by the problem, we come to better understand the problem -- and the aspects of it -- that need to be solved. To do this effectively, we should interview stakeholder representatives to understand how they are affected by the problem, and what they would like to see in the solution.

In addition, we need to be aware of the benefits of solving the problems identified, so that we can evaluate the cost-benefit of various solutions. The benefit of solving the problem is of course independent of the cost of the potential solutions. Nevertheless, knowing the value of solving the problem can help us to establish a ceiling on how much we would be willing to spend to solve it. This helps us eliminate some potential solutions early in the identification process, before we spend a lot of time describing the potential solution.

One potential trap in the Problem track is to "problemify" a solution (as one client with whom we have worked put it). In other words, someone on the team already has a solution in mind and
merely rephrases it as a problem. We need to make sure that we understand the real problems, not just reinforce our preconceptions of a solution. This is more difficult than it may first appear -- it is very easy to jump to conclusions about the problem. At the same time, we cannot become paralyzed by endless problem analysis.

**The Solution track**

In the Solution track we explore alternative solutions to the problems identified in the problem track. Projects usually fail here by not being creative enough -- not exploring enough alternative solutions, focusing on defining the details of a single solution too early. We can use various brainstorming techniques to identify a number of potential solutions to each problem identified. The focus should first be on identifying (without judgment) potential solutions, excluding only solutions that are obviously too expensive given the value of solving the problem.

In identifying solutions, we should make sure that the solution will help each of the types of users affected by the problem. The people affected by the problem become one set of stakeholders to the solution. Other stakeholders include people who are affected by the solution (people who will support the solution but might not use the solution), and the project team itself.

Use cases can help us to define potential solutions by helping us to describe what they will do for the stakeholders. They help us to explore and define the behavior of the solution. It's important to recognize that a use-case model describes a potential solution. As a result, it may be useful to develop several separate use-case models to compare alternative solutions. These models can help us to visualize different solutions and get feedback from stakeholders on different proposals, all of which can help us decide whether to move ahead with one of the alternatives.

When developing these use-case models we at a minimum identify the primary actors and briefly describe and outline the alternative flows. We will need this much information to define the scenarios and develop a rough estimate of the solution's overall schedule and cost. Then we can decide whether or not to proceed with the solution. As we describe the solution using use cases, we need to walk through the model with key stakeholders in order to confirm that we correctly understand the problem, and to evaluate if this potential solution would meet the stakeholders' needs. Their feedback mitigates the risk of spending lots and lots of money developing something that nobody wants and that doesn't solve the real problem. Sadly, this happens more times than we would like to admit.

In addition to the use-case models, small functional prototypes are often useful to gather feedback on potential solutions. For instance, we might walk the stakeholders through a set of scenarios using screen mock-ups to get feedback on proposed solutions. The important thing is to keep the investment low and flexibility (being able to revise the solution approach) high while our idea of what we are going to build is still evolving.

The Vision document presents a summary of a particular solution and the perceived problems that it solves. If we have several credible potential solutions, then we should develop separate Vision documents for each one. Separate vision documents are valuable if we want to make a
formal choice between alternative approaches. Typically, though, most projects settle on a single approach after an informal exploration of alternatives.

The Project Track

The development team performs a third set of activities in the Inception phase -- those related to planning and initiating the project itself. While the Solution track activities identify and define potential solutions, the function of the Project track is to estimate the cost, resources, and a high-level schedule for the project, to determine whether the proposed solution is economically viable.

The fact that we lack defined solutions at the start of Inception creates a dilemma: at the start of this phase (and therefore the product), we really don't have a clue about how long the project will take and how much it will cost. This is the fatal flaw of the "plan at the beginning" crowd who want to devise a very detailed plan early on and then execute to the plan for the rest of the project. We have to accept a fair degree of uncertainty in the Inception phase in order to move ahead.

Once we do select a potential solution, however, our work on the Project track creates a plan for the rest of the project:

- Define the duration and goals of each phase
- Plan the iteration structure of each phase
- Map preliminary risks and scenarios onto iterations
- Create a resource plan for the project
- Identify the skills and staffing levels that will be required for the phases
- Develop an overall project estimate

Occasionally, a question arises regarding whether we need multiple iterations in the Inception phase: the answer is, "sometimes". In the following cases, it may take more than one iteration to determine whether a particular solution is really the right one:

- The business problems are difficult to diagnose
- The suitability of a solution is hard to understand
- The solution is technically unproven
- It is hard to choose between multiple solutions

It is usually impossible to predict this at the very start of the Inception phase, so the decision is usually made after completing the first iteration. At this point, we can decide whether or not we need more time to assess the viability of one or more proposed solutions before proceeding to the Elaboration phase.

The Inception Phase for the ACME Super ATM

Going back to the example, if we're going to build this new super-duper ATM (Figure 12) -- Java-based, more modular and with a lower cost of development, more configurable so we can do all kinds of different dispensers (postage, event tickets, pay parking tickets and so on)-- then we need a framework for planning the project.
Since as we discussed earlier the purpose of the Inception phase is to bring business risks under control, we should assess the market need. Perhaps we can define this need as giving people flexible access to their funds and enabling them to use these funds in a number of different ways. We are building this ATM for financial institutions, so we have identified a target market. To complete our work in the Problem track, we will need to gather a set of stakeholders from those institutions to understand their needs. We will also need to have customers represented by other stakeholders who we will work with to make sure that we understand the problems that they would like our potential solution to solve.

In the Solution track we take this problem description information and formulate ideas about how we will solve it. In fact, we have jumped ahead a little bit because we appear to have started out with a solution in mind (a new kind of ATM), but in practice this is rather typical. Since we do have a solution in mind, we will need to figure out whether there is actually a need for this in the market. One way to do this is by comparing the problems experienced by stakeholders with existing solutions on the market. If we find that there is no competition, our task will be easier. If other competitors have partial solutions to the problem, then we need to understand how our solution could be different. We capture all this analysis in the Vision document for the solution.

One of the risks of building this new ATM is that the market is crowded. Who wants to buy this? If it doesn't significantly differ from existing products on the market, then why do it?

Our potential solutions will encompass various “what if” scenarios. For example:

- If we cannot price the product aggressively, maybe we can build a super-duper ATM.
- If it turns out that our new fancy ATM is four times more expensive than the existing ones on the market, and the financial institutions make a fraction of a cent per transaction dollar, maybe we will have a great solution that no one wants to buy.

We will have to assess our total cost of development, divide it by the number of units we expect to ship, and compute the development cost per unit. We then need to decide how much margin we will need to cover all the other things we need to do to sustain the product -- technical support, marketing, and related overhead.
In addition to cost and pricing information, we need to assess the real market need -- do customers really want the kind of services offered by the ACME Super ATM? Will people use an ATM to pay bills, get theater tickets, and perform non-traditional ATM services? After all, when ATMs were first introduced, it took many years before the public readily accepted them.

To try to answer these questions, we need to build a small prototype of the potential solution. For instance, we could mock-up our new ATM and subject it to test marketing, or we could build an ATM with limited functionality and take it to a usability lab, bringing people in to see if they would want to use its services. Think how much money the "new economy" firms of a few years ago could have saved had they done this kind of analysis before spending millions perfecting business models around services that people did not want to purchase.

Building this prototype also has an additional benefit -- it helps us to understand the potential costs of the solution. As noted previously, this information feeds back into the estimates needed to evaluate the financial worthiness of the solution.

**Concluding the Inception Phase**

To conclude the Inception phase, we need to do a few more things. First, we need to settle on a single solution proposal that we are going to take forward. To do this, we need to make sure that the solution:

- Solves the right problem
- Is (as far as we can tell at the moment) technically feasible
- Is economically viable

We need all stakeholders to agree on these three points, at least insofar as they are willing to take the project to the next step, assessing the technical viability and developing the "architecture" of the approach.

Second, we need to create the plans for the project, laying out the phases, describing major milestones, and mapping risks and scenarios to iterations. We need to do this to assess the economic viability of the project, but we also need to do this to move the project ahead.

Finally, we need to start gathering the team for the rest of the project. If the project is to go forward, we are going to need people with specific skills to fill out the resource plan, and it might take a while to find the right people.

It should be said at this point that a completely acceptable, and perhaps even desirable, outcome of the Inception phase is a decision *not* to go ahead with the project. Some problems are not solvable given the investment that the organization can afford to make, and other problems cannot be solved at all. It is sometimes better to decide to focus on solving other problems than to waste time solving a problem whose solution is uncertain or expensive. Canceling a project at the end of Inception is often the best solution; it is a much bigger (and more expensive) mistake to let a project continue when there is little hope for real success.
Planning and Managing the Elaboration Phase

At the conclusion of the Inception phase we have achieved a couple of important things: we have agreement on:

- The problems being solved
- A tentative solution to those problems
- How much that solution will cost

As we move into the Elaboration phase we turn our attention to assessing the technical details of the solution, ensuring that the solution is really technically feasible, and making important and irrevocable technical decisions about the solution. To do this we have to come to a more detailed understanding of the requirements, and we have to build and test critical parts of the solution to make sure our assumptions are correct.

As we have discussed already, the basic pattern of iteration planning is to choose a set of scenarios (along with related supplementary requirements) using a set of risks that we need to address in that iteration (as defined by the Project plan). The risks addressed in the Elaboration phase are technical risks, or risks related to technical aspects of the solution. Often, these technical aspects relate to various system qualities such as performance, reliability, scalability, security, portability, and so on. We say that the Elaboration phase focuses on architectural issues because solving these "architectural" challenges requires a consistent, systematic series of decisions and actions that has a pervasive effect on the solution. Changing these decisions later, after the development of the system is well under way, is typically prohibitively costly in terms of both schedule and budget.

The Elaboration Phase and the Art of Scenario Selection

Planning the Elaboration phase is an art that requires a certain amount of intuition about the kind of technical challenges that the team will encounter when forced to analyze, design, develop, and test a particular scenario. A scenario consists of the basic flow of a use case plus zero or more alternative flows. The architecturally significant scenarios are those which are going to strain the solution to its limits. Intuition comes into play at the point when planning needs to be done, because at that time you have very little in the way of definition to work with -- the scenarios may be loosely identified, but the use case and their flows (on which the scenarios are based) have not yet been fully described. Furthermore, since we don't yet know the details of the use cases we certainly can't know how the system will realize those use cases.

The art of choosing the right architecturally significant scenarios lies in being able to visualize and anticipate that a particular scenario will entail a particularly difficult technical problem. This anticipation is honed by experience with similar problems, and requires insight into a variety of different problem-solving approaches. The skills required are often those of both generalist and specialist -- the ability to broadly understand a large variety of approaches combined with the ability to reason deeply about specific approaches.

In selecting scenarios for the Elaboration phase, we have to strike a balance between:
• The number of scenarios
• The degree to which they force us to consider technical risks
• The amount of time and resources we have available

If we selected all scenarios we would cover most of the technical risks, but we would also take up all available time on the project. If we select only a few scenarios, we risk missing some significant issue. The art of scenario selection also lies in being able to address the greatest number of risks with the fewest number of scenarios necessary to minimize the risk that we may miss something.

One set of scenarios that are important to consider but easy to forget are those related to installation, configuration, and maintenance of the solution. Today, these issues represent some of the more complex aspects of many solutions, because there is never a good time to shut down a 24 by 7 system for maintenance. As a result we may need to incorporate capabilities that allow us to make changes or upgrades without having to shut the system down. These capabilities require great forethought, planning, and creativity, and we should handle these issues now in the Elaboration phase.

The best way to achieve this balance is to have an experienced solution architect. By experienced, we mean someone with the required breadth to understand the potential for alternative approaches and the depth to understand the risks inherent in those approaches. Finding the right person for this role is often the project manager's most important task in the early days of the project.

The number of iterations needed in the Elaboration phase will depend on several factors:

• The total number of architecturally significant scenarios
• The number of scenarios that can be worked simultaneously
• The dependencies between issues that you will address as the scenarios are worked

The team in the Elaboration phase is still likely to be rather small to keep communication overhead low, so it may take several iterations to prove the viability of the solution.

Assessing Technical Feasibility and Architectural Stability

The important idea about architecting is that we need to build and test parts of the system in order to be sure that we have an acceptable solution. As with scientific experiments, a proposed solution is a kind of theory that we need to prove or disprove. Like scientists, we need to conduct experiments to see whether our theory about the solution's suitability is correct. We perform these experiments by building part of the system and conducting tests. We don't need to build the whole system to conduct these tests; we can build just enough to test our hypothesis about the suitability of some aspect of our solution. Another way of thinking about the Elaboration phase is that it is for R&D, as opposed to the Construction phase, which is more like manufacturing.

More often than not, assessing the technical feasibility of a proposed solution means assessing its performance and scalability. It may seem that we have to build quite a lot of a solution in order to evaluate its suitability, but the reality is that in many systems the number of parts that are affected by scalability concerns is rather small. As a result, our challenge becomes selecting the parts of
Driving Iterative Development With Use Cases

The system that will be affected by various technical challenges (architectural risks), building them into an architectural prototype, and then subjecting them to simulated real-world workloads.

**The Elaboration Phase for the ACME Super ATM**

First, we need to gain agreement on capabilities by identifying all of the primary use cases for the ACME Super ATM. In practical terms this means that we need to get everybody to agree that a customer should be able to withdraw cash, deposit funds, transfer funds, and all the things you would expect from a conventional ATM, plus novel things like paying bills and purchasing theater tickets. We won't worry about the structure of the model or things like inclusion or extension, or generalization of either use cases or actors. We simply want to make sure we really understand all the major things the system will have to do. In addition, we need to outline the basic flow and identify the alternative flows of each use case so that we have a general understanding of its scope.

The next thing we need to do is to identify the architecturally significant scenarios. For the ACME Super ATM, many of the architecturally significant scenarios arise not from the functionality of the ATM, which is fairly simple. Rather, they arise from the alternative flows related to system failure, security and recovery, and from the need to run a large number of transactions in parallel on the ATM network. We document these architectural scenarios in our Software Architecture Document.

In order to assess the scalability of the proposed solution, we are going to simulate running 1,000 simultaneous withdrawals of cash. We also need to evaluate the security of the architecture of the proposed solution by simulating physical tampering in an attempt to rob the ATM. We will also truncate the network connection during a cash withdrawal, and then cut power to the ATM in the middle of a transaction, to test the fault tolerance and failure recovery capabilities of the system.

For example, if we indicate that we want to withdraw a particular amount but lose power before the actual money is dispensed, can the transaction be rolled back? Let's also make sure that if we dispense the money but lose power before the transaction is actually completed with the bank that it will be completed once power is restored.

We should also do some work on configuring new kinds of transactions to ensure that we will be able to do things other than dispensing cash. To evaluate this, we could design, implement, and test the scenario in which we print a theater ticket instead of dispensing cash.

For each of these scenarios, we create a use-case realization, which is basically a sequence or interaction diagram that identifies the components we are going to use to fulfill the behavior specified by the scenario so that we can allocate functionality out to them. At the same time, we are going to develop test cases that validate the scenarios so that we know what the desired behavior is.

It's worth spending a moment considering what we mean by testing in the Elaboration phase. In conventional thinking, testing (usually performed in a later phase) primarily focuses on validation: making sure the system behaves as specified. When we test in the Elaboration phase, though, we are really trying to uncover problem areas that we need to work on, rather than validating that...
the behavior is correct. As a result, we want to focus more on load testing, performance, and throughput, and not so much on stability and low defect counts. In addition, we don't want to spend any amount of time on testing the user interface at this point, since that will change greatly over the remainder of the project.

**Managing Scope Changes in the Elaboration Phase**

We typically end the Elaboration phase having identified more work than we started the phase with. The reason for this is simple -- we have now had some time exploring the requirements to find out what is *really* needed to solve the problem, and this usually means there is more work to do than we initially thought. From a planning perspective, if the project was already tightly scheduled or budgeted, this means that one of several things may need to happen:

- Extend the schedule
- Extend (increase) the budget
- Extend both the schedule and the budget
- Scale back the scope of the solution

Extending the schedule, the budget, or both may be the right thing to do. If the problem is valuable enough to solve, or perhaps mandatory to solve, then asking for additional time or budget will likely be met with little resistance, provided that the new estimates are credible. The work we have done so far will help here: since we now have built and tested the technically complex parts of the solution, our certainty that we can build the solution has increased significantly. In addition, building parts of the solution also reduces the amount of work remaining (that we need to estimate), further reducing the estimation error. The project going forward is much easier to estimate than the part of the project that is behind us.

Still, there may be reasons why we should cancel the project at this point. It is obvious that if the proposed solution is technically infeasible then we should not continue the project. Sometimes this happens, but it is more likely that we find that the solution is significantly more expensive than we had originally hoped. The decision at that time is whether we should increase the project funding and continue. We should base this decision on the threshold for maximum project funding -- which was established back in the Inception phase when we considered the value of solving the problem -- and its close cousin the maximum amount we would be willing to spend to solve the problem.

If we find that the cost of solving the problem exceeds the benefits of solving the problem, then the option that remains for us is to reduce the scope of the solution. This typically involves some fairly intense negotiation between the project manager and the stakeholder representatives (people representing the groups affected by either the problem or the solution) to determine what can be cut. This negotiation process is typically iterative as well -- removing requirements often has unexpected effects on the cost and schedule estimate. In addition, when eliminating a particular requirement it is often difficult to determine the cost effect without doing a fair amount of investigative work, perhaps even building some small prototypes to assist in the estimation.

Likely candidates for scope reduction are alternative use case flows. You will rarely be able to eliminate entire use cases, but you might determine that some of the alternative flows are not
needed or are so unlikely to occur that they need not be supported. Many teams try to manage scope at a "feature" level, but the lack of a clear definition of what a feature is, as well as the fact that a feature may cut across (and effectively invalidate) use cases, makes them relatively useless for scope management purposes.

Concluding the Elaboration Phase

At the end of the Elaboration phase we need to be rather unyielding on the issue of mitigating technical risks -- if we have mitigated them then we are ready to proceed to the Construction phase. If we have not, however, we should not fool ourselves into thinking that we are ready to move ahead. Remember, once we reach the Construction phase we have gone down a path that we cannot easily retrace without potentially throwing out lots of work. The impact on schedule and cost of doing this is likely to be severe, potentially putting the project at risk. We should not make the decision to conclude Elaboration lightly.

Coming back to our ACME ATM example, let's see what this would mean. At the end of the Elaboration phase we need to have a stable, proven architecture that can deal with the technical risks we identified earlier, as well as any new risks that we have discovered during the Elaboration phase. This means that we have tested scenarios in which we have tried to break into the system, caused the network to fail, and disconnected the power, as well as running our most likely transactions with thousands of simulated users. If we have done all this successfully, then we should be feeling pretty good about the viability of our project.

In addition, because we know a lot more about the solution we ought to have much better confidence in our estimates by now. This is not only because we have concrete information upon which we can base estimates (such as function point counts), but also because we have more information in the form of real code (which we can start driving through a CoCoMo II model to cross-correlate our function point-based estimate). Furthermore, we have identified all the use cases -- which means that we are not going to discover any major new things the system will need to do -- and we have the major architectural scenarios detailed. We also have all the architecturally significant supplementary specifications documented. We should have completed test plans for these requirements and scenarios, and we should have a partially completed design model, test cases, and executable code. In short, we have a lot of the important work out of the way. All this means that we have a very stable basis for the remainder of the project.

Figure 13 shows our overall progress to date.
Planning and Managing the Construction Phase

The focus of the Construction phase is primarily on managing the logistical details of getting the rest of the work done in the time allotted. With the solution architecture in place following the Elaboration phase, it is common to see the size of the project team increase to allow more work to proceed in parallel. The rest of the work follows a pattern similar to that of the Elaboration phase: the remaining scenarios are detailed, analyzed, designed, implemented, and tested. In the Construction phase we have to implement all functionality, including going back to technically simple code that we stubbed-out in the Elaboration phase. As we implement the rest of the functionality, we also need to continue to run the architecturally oriented test cases from the Elaboration phase to ensure that new work does not degrade or destabilize the architecture.

The testing work changes character as we enter the Construction phase. Up until this point we have focused on proving the suitability (Inception) and technical feasibility (Elaboration) of the solution. Our testing has reflected that -- we have not really tested the user interface, although we have done extensive usability design work to ensure that the visual style and metaphors used in the application are stabilized. With this work behind us and the GUI stabilizing, we can start GUI testing in earnest. To reduce the ongoing cost of running these tests we will want to automate most of the GUI tests so that we can run them after every build.

In addition to this development work, it is now time to start writing documentation and support materials, and to start planning the rollout of the solution; the actual rollout will occur in the Transition phase, but the planning work needs to start now.

Managing Scope in the Construction Phase

In addition to the scope management strategies discussed above (Elaboration), there are others that you can pursue if there appears to be too much work left to do. One strategy that might reduce the amount of work while not greatly increasing risk is to reduce the formality of work associated with some of the simpler scenarios. In practical terms this means that we can make a conscious choice to merely outline (and not detail) simple scenarios that are already well understood. It might
also mean that we decide not to do formal analysis and design using modeling techniques, but instead use rapid prototyping tools to build these simpler system parts. Behavior that is a good candidate for this strategy tends toward simple data management:

- Add, modify, and delete
- Simple menus and navigation
- Behavior governed by simple business rules

Taking this approach enables us to conserve valuable time for scenarios that implement complex business transactions in which making the wrong decision would be very costly.

**Strategies for Managing Change**

It is common for the scope of the project to increase throughout the Elaboration phase and into the Construction phase. This is not necessarily bad -- it results naturally from understanding more about the real problem being solved and the cost of potential solutions. A problem arises because the schedule and budget for the project do not typically expand in tandem with the scope. In addition, if you do not properly manage expectations, they do not adjust to the increase in scope. Therefore, many projects fail because they take on the additional work without getting buy-in from all the stakeholders. The result is a perception that the project has failed (to meet budget and schedule, at any rate) even though the project team thinks they have done the right thing.

This problem is both easier and harder to solve than it sounds. The easy part of the answer is that solving the problem is a simple matter of getting the stakeholders to agree on what is important, including whether achieving the new scope warrants changes in budget and schedule. The hard part is that this is not that easy -- projects typically have a diverse set of stakeholders that do not necessarily share the same objectives.

A couple of things can make this easier. The first is establishing at the outset an expectation of the fact that the scope of the project is going to change, and that you are going to have to work through the issues of scope, schedule, and budget. The second thing that we need to do is to get all the stakeholders to agree on a process for how decisions about these issues are going to be made. Getting agreement on this early makes it easier to have the discussion later on.

**Concluding the Construction Phase**

By the end of the phase, we will have implemented all the functionality and will be ready to start rolling out the application to its users. As discussed previously, this does not necessarily mean that we ended up where we thought we would -- but if we have been successful in managing risks and scope it means that we have made the trade-offs necessary to deliver the right solution within an acceptable time frame and budget.

If we have in fact met our goals of a "good enough" solution, it is important that we know when to stop. It is tempting to throw in "just one more feature", but we need to resist the temptation. Adding things now will very likely delay the project and may destabilize the delivery of the solution. Furthermore, keep in mind that most systems are too complex and feature-packed to be really
useful. In many cases we would be better off reducing the number of features, making the essential scenarios work very well, and ensuring that the system is easy to use.

**Planning and Managing the Transition Phase**

If we have done our job in the preceding phases, the Transition phase is focused on delivering the solution to its intended users. There is still some activity in each of the disciplines, however, with most of the effort focused on **deployment** and **fixing defects**. Moreover, while we should not plan to be implementing new requirements at this point, there will inevitably be requirements (small in scope if we have done a good job managing them so far) that are discovered, analyzed, designed, implemented, and tested. There will also, naturally, be defects to fix.

At this point, however, it's not always all about development. Most significant systems have significant operational aspects that we have been working to sustain throughout the project. Now is the time to move the application from a development environment into a production support environment. This means that we need to train support staff, establish operating procedures, and convert production data before the solution can be considered **complete**.

Furthermore, very few systems are so intuitive to use that they require no training. In fact, many of today's solutions involve not only new software and hardware, but also changes to business processes. During the earlier phases we should have planned, designed, and tested these business process changes, and now it is time to put them into place (along with the system changes) to support the new business model. We will have drawn on our use cases to develop appropriate training and documentation, and now we can take advantage of that work.

**The Transition Phase for the ACME Super ATM**

In our ACME Super ATM example, the kinds of things that we would need to do relate mostly to the logistics of getting the software installed on the ATMs in the manufacturing operation, deploying these machines out to the appropriate locations, and getting them configured and running on the ATM network. The actual roll-out of systems will continue over a period of years, and will represent a whole program of its own, so our principal concern is on the first wave of deployments to make sure that the ongoing deployment efforts will be smooth.

**Concluding the Transition Phase**

As we have done at the end of each iteration and each phase, we need to take time to evaluate our results in the Transition phase. In addition, since we are at the end of the project, we also need to take stock of the overall project. Even if the project has fallen short of its goals, and maybe especially if it has fallen short, we need to take time to understand what has gone well and what we need to work on in future projects. It is natural to want to move on from the project at this point, particularly if it has been a success. Taking time now may help us later, and we need to make it a priority.
Summary

In this article we have walked through each of the phases of a project, focusing on their purpose in the broader project. We have considered the risks addressed in each phase, and discussed strategies for dealing with these risks. Throughout the project, we have seen how use cases hold the project together by focusing our attention on how the stakeholders are going to use the system.

We have also discussed how different estimation approaches help us to plan and manage the project. In every phase we need to provide estimates and manage resources and risks, although the work is slightly different in each because each phase has a different risk focus. We discussed how iteration and phase-end milestones are used to evaluate progress and keep the project on track.

An iterative approach to developing solutions is a powerful tool that helps us to bring predictability and responsiveness to solution development efforts. Using these tools gives us an edge in delivering the solutions that our stakeholders expect of us, in a timely and cost effective manner.

Footnotes

1 A technique used to implement technically simple parts of the system superficially by hard-coding a default response to a function call. As an example, a function isValid(someObject) could be stubbed-in by returning a 'true' value instead of actually performing the validation. We would use this technique when the implementation of the function will not affect the evolving architecture of the system. By postponing implementation of this function, we free ourselves to focus on more challenging aspects of the system's architecture.

2 Adapted from Walker Royce, Software Project Management: A Unified Framework, p. 276.

3 "Constructive Cost Model", see http://sunset.usc.edu/research/COCOMOII/

4 This is one of the implicit tenets of approaches like eXtreme Programming, in which the "user representative" is fully responsible for communicating the business problems and objectives—even much of the solution definition-leaving the development team to focus purely on development issues. In practice, we have found that it is important for the development team to understand the business problem so that they can participate in finding creative solutions. This approach has a price—technical teams must become much more involved in understanding the business in order to be successful, and they must consciously accept this new responsibility.

5 One customer with whom we have worked created a serious problem for themselves by locking in the schedule and funding for projects before they even understood the problem or potential solutions. Project managers' compensation was put at risk if they did not meet the arbitrarily determined cost and schedule targets: needless to say, nearly every project came in on time and within budget. However, since the projects were created with very little idea of what needed to
be delivered, the deployed solution often failed to produce meaningful business results. In this case, the project funding and performance measurement model, instead of improving results, was actually a strong contributing factor to poor solution results.

6 Nothing is absolutely irrevocable, but what we mean here is this: we are going to make decisions that, if reversed or changed, are going to threaten the viability of the project. In other words, we had better do our homework to make sure that these decisions are correct.

7 The maximum amount we would be willing to pay to solve a problem takes into account the required rate of return for projects of a similar risk, and is essentially the net present value of the benefits less the costs of solving the problem.

8 A feature really has no consistent definition; it is basically some capability of interest to someone. The lack of clear definition of what is and is not a feature is one of the reasons why features are not very useful for scope management.

9 There is a famous observation (dubbed Brooks' Law after Fred Brooks, who wrote about it in his seminal work The Mythical Man-Month) that says that adding people to a late project makes it later. This is true in cases where the architecture is not stable and work cannot be effectively partitioned into independent units that can be performed by semi-independent teams. In these cases, the overhead of teams that have to constantly communicate to avoid conflicting with one another actually slows down the work of everyone, resulting in less work getting done even though more people are working on the project.

© Copyright IBM Corporation 2004, 2006
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)