Managing iterative software development with use cases

Ian Spence, Technical Team Lead, IBM
Kurt Bittner, Communities of Practice Architect, IBM

21 Jun 2004

from The Rational Edge:

Although the Rational Unified Process,® or RUP,® provides a basic framework for iterative development, it does not tell the practitioner exactly how to run an iterative project. This article begins filling in the blanks by providing a brief introduction to the iterative lifecycle RUP promotes. It explores the forces that shape the project lifecycle and the role that use cases play in controlling these forces.

The Art of Iterative Management

In our book Use Case Modeling,1 we explain that use cases are a practical technique for documenting requirements: They provide a way of discussing system users' goals and how the system works with users to achieve these goals. As a result, use cases are also helpful to people who ensure system usability, document the system's behavior, and design, develop, and test the system.2

Even though use cases provide all these benefits, however, there can be confusion about how to work with them iteratively, or, even more important, how to drive iterations with use cases. Applying the concepts in an iterative environment can be quite challenging, as the disciplines of planning, requirements gathering, analysis, design, implementation, and testing are interleaved in a way that provides no clear end to the activities of one discipline and the start of the next. This can be disorienting — so much so that many teams who think they are following an iterative approach are actually working in a "waterfall" style while using the terminology of iterative development.

The reality is that we do a little from each of the disciplines in each iteration. The art of driving iterations lies in knowing what to do and when, how much from each discipline to do, and how to evaluate progress. Iterative development offers great promise in terms of reducing risk and improving time to market, but it requires changes at once subtle and dramatic in the way that projects are run.

To understand these changes, we must first understand the entire lifecycle of an iterative project. Let's begin by looking at what we mean by an iteration.

What Is an Iteration?

An iteration is a division of time in a project. In his book Software Project Management: A Unified Approach,2 Walker Royce defines an iteration as:

A distinct sequence of activities within a single phase, resulting in a release; includes a well-defined plan and a documented result.

We divide projects into iterations to gain greater control over the project, and to mitigate risk. To ensure that we are making progress, we force each iteration to produce something tangible: a "release." This release may be:
A prototype we use to demonstrate some capability.
An "internal" release that we will use as a basis for further development and testing.
An "external" release that we ship to customers in some form.

To produce a release, the iteration will involve applying, at least in part, the disciplines of project management, requirements, analysis and design, implementation, and test. Since projects typically have multiple iterations, the project will need to apply these disciplines over and over again as the project progresses. It is this continuous re-application of the disciplines that led to defining these process segments as "iterations" and the entire approach as "iterative development." Over the course of the project, each iteration moves the project closer and closer to its goal in a deliberate, intentional manner, removing different risks at each iteration.

Avoiding Pitfalls
This deliberate march toward delivery is the hallmark of true iterative development, and what separates it from "random incremental" development, in which small parts of an application are developed in a relatively unplanned manner. In planning iterations, we specifically want to avoid:

1. Using the iterative nature of the project as an excuse to never finish anything. In other words, the approach should not become a charter for procrastination that sanctions a "Don't worry — we can finish that in the next iteration" attitude.
2. Allowing the results of an iteration to break or compromise results achieved by previous iterations.

As Ivar Jacobson points out in The Unified Software Development Process, the iterative lifecycle is not:

- Random hacking
- A playpen for developers
- Something that affects only developers
- Redesigning the same thing over and over until developers finally chance on something that works
- Unpredictable
- An excuse for failing to plan and manage

The Rational Unified Process: A Framework for Iterative Software Development
Based on many years of guiding iterative customer software development projects, Rational experts evolved a set of basic rules for working with iterations:

- Every iteration should produce an executable release (either internal or external) of the software. The release should fulfill more of the project's requirements than the previous release (from the last iteration). This increase in delivered requirements is called an increment.
- Iterations are defined by their intended results as well as the evaluation criteria for these results.
- Every iteration should actively address and reduce project risk.
- Every iteration should be treated as a discrete time box.
- During an iteration, the project team should focus on meeting the objectives of that iteration alone, doing whatever they can to ensure the iteration's success, and no more.
- The results of every iteration should be objectively assessed; the team should be prepared to rework the solution and project plan as required.

Although these guidelines are useful, they are not detailed enough to help people plan a particular iteration or decide what the objectives of the iteration should be. How should we decide which project risks to address first, and which requirements to implement within which iteration? That is why, to help teams effectively plan and manage iteration-based projects, Rational created a control framework that guides them in establishing appropriate objectives for their iterations and allows them to objectively assess project progress with each iteration.

That control framework is the RUP. It has four sequential phases, each culminating in the achievement of a major project milestone. These phases and milestones (Table 1) provide guidance and a roadmap for project planning and control.

Table 1: RUP Phases and Their Milestones

<table>
<thead>
<tr>
<th>RUP Phase</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td>Lifecycle Objectives (LCO)</td>
</tr>
</tbody>
</table>
What Are Phases and Milestones?

In his book *Software Project Management: A Unified Approach*, Walker Royce defines a phase as:

> The span of time between two major milestones of the process, during which a well-defined set of objectives is met, artifacts are completed, and the decision is made to go on to the next phase.

and a major milestone as:

> System-wide event held at the end of each development phase to provide visibility to system-wide issues, synchronize the management and engineering perspectives, and verify that the goals of the phase have been achieved.

To be able to effectively use the RUP milestones as an aid to planning and controlling our iterative projects, we need to understand the goals and objectives of each phase, and how these goals are verified by reaching the major milestone at the end of the phase.

Before we dive into the specific phases and milestones of the RUP, however, it's worth discussing why we even bother with phases. Some approaches (such as Extreme Programming) do not use phases at all — anything can happen at any time.

The Importance of Phases

We use phases as ways to focus on, and therefore mitigate, certain kinds of risks. At the end of each phase, we should have mitigated a particular class of risks that, if left unmitigated, would potentially cause serious problems if we were to continue. It is easier to understand the purpose of a phase if you know the risks it addresses. Table 2 presents the primary risks for each RUP phase.

**Table 2: RUP Phases and the Risks They Address**

<table>
<thead>
<tr>
<th>RUP Phase</th>
<th>Addressed Risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td>Business Risks</td>
</tr>
<tr>
<td>Elaboration</td>
<td>Architectural/Technical Risks</td>
</tr>
<tr>
<td>Construction</td>
<td>Logistical Risks</td>
</tr>
<tr>
<td>Transition</td>
<td>Solution Roll-Out (Delivery) Risks</td>
</tr>
</tbody>
</table>

With the successful completion of each phase, the project team retires a set of risks and can move the project safely forward. The transition from one phase to the next represents a significant business decision and vote of confidence in the project. It does not represent the completion of any specific activity or artifact. The success of a phase is judged by how successfully its milestone has been achieved.

From a management perspective, phases provide a way of assessing project progress, so we can ensure that the project is actually steadily progressing toward the delivery of a high-quality product, and not wandering aimlessly through an infinite series of iterations.

The Importance of Milestones

Milestones serve many purposes; most critically, they provide concrete evidence of development status for stakeholders who have to make certain crucial decisions before work can proceed to the next phase. Milestones also enable management, as well as the developers themselves, to monitor the progress of the work as it passes key points in the project lifecycle, so they act as a series of checkpoints for the project as a whole. Each major milestone provides an opportunity to synchronize stakeholder expectations, assess project progress, and decide whether to continue the project. It is essential that the team treat milestones seriously — as a series of gates through which the project must progress in the defined sequence. For example, regardless of what phase the schedule says the project should be in, if the initial Lifecycle Objectives milestone has not been met, then the project is still in the Inception phase. We will discuss milestones in greater detail later.
**Iterations, Phases, and Milestones**

The definitions of *phase* and *iteration* are often confused. Both are periods of time that result in the achievement of a set of clearly defined objectives. There are two major differences:

1. The phases and major milestones are defined by the process, and they are common to all projects. The number and size of the iterations, and their specific objectives, are at the discretion of the project manager and the development team.
2. The phases are not time-boxes in the same way that iterations are. Iterations should be concluded when the time runs out; phases are concluded when their milestone has been met.

Each *phase* consists of one or more *iterations*, with the completion of each iteration representing a *minor milestone* for the project and contributing to the successful achievement of the phase's objectives. Figure 1 shows the relationship of each set of iterations to its respective phase, the relationship of phases to one another, and the major milestones that mark the conclusion of each phase. **Note:** If a project team manages to achieve the objectives of a phase in a single iteration, then the difference between the phase and the iteration would be negligible.

![Figure 1: RUP Lifecycle Phases and Iterations](image)

**Figure 1: RUP Lifecycle Phases and Iterations**

Figure 2 shows the most common view of the RUP *lifecycle*, presenting the *effort* typically expended on each process *discipline* in each of the *phases*. This is a valuable view of the process *dynamics*, but it needs to be complemented with a view of the process *objectives* as expressed by the goals (discussed below) and *milestones* (also discussed below) of each phase. Without this additional understanding of the goals of the phases and the achievements represented by the milestones, it is impossible to know why, when, or to what purpose the disciplines are being applied, or if they have been applied effectively.

![Figure 2: The Rational Unified Process Effort Profile](image)

**Figure 2: The Rational Unified Process Effort Profile**

**Goals of the Project Phases**

A phase is a period of time, culminating with a major milestone, during which the project team addresses a particular set of risks. The
RUP phases help teams assign purpose to iterations, and to focus on particular kinds of issues that must be addressed before the project can proceed. *Successfully concluding a phase means that you have mitigated the designated risks and resolved the issues so that the project can successfully proceed to the next phase.*

### Inception Phase Goals

The Inception phase has the following goals:

- Identify and mitigate business, project, and funding risks.
- Assess the viability of the project, both technically and financially.
- Agree to the scope and objectives of the project.
- Form an overall plan for moving ahead.

In Inception, the main focus is on mitigating the risk that the project may be either economically undesirable or technically infeasible. For this, the team must explore the benefits and costs of the project so that a firm decision can be made about whether to proceed at the end of the phase.

Inception culminates in the **Lifecycle Objectives (LCO) Milestone**, which entails an assessment of the viability of the project as a whole. The LCO milestone is successfully achieved only when stakeholders agree that the project is feasible — and that the business case for the project is achievable if the project proceeds as planned. Everyone should also concur that the project is worth doing and that time and cost estimates are credible.

### Elaboration Phase Goals

The Elaboration phase has these goals:

- Bring architectural and technical risks under control.
- Establish and demonstrate a sound architectural foundation.
- Establish a credible plan for developing the product.

Elaboration's main objectives are to prove the architecture to be used in the product development (by verifying that the proposed architecture will support the solution) and to eliminate the project's highest-risk elements. At the end of Elaboration, the project manager is in a position to plan the activities and estimate the resources required to complete the project.

If a team does not understand the significance of a sound architecture, they are likely to misinterpret the purpose of Elaboration and treat this phase as a time to "elaborate" the requirements — in other words, to define detailed requirements.

Elaboration culminates in the **Lifecycle Architecture (LCA) Milestone**, which involves proving and baselining the solution's architecture. The project team achieves the LCA milestone successfully when they can demonstrate an executable architecture (typically one or more architectural prototypes) that meets key architectural requirements, is able to support the vision for the product at a reasonable cost, and can be produced within a reasonable amount of time. Milestone assessment is based on the stability of the requirements, the understanding of the project's risk profile, the credibility of the plans and, most important, the results of testing the architectural baseline.

### Construction Phase Goals

The Construction phase has these goals:

- Bring logistical risks under control.
- Develop the solution.
- Ensure that the solution is ready for delivery to its user community.
- Achieve adequate quality as rapidly as is practical.

This is where most of the "heavy lifting" of the project gets done; it is where the bulk of the work occurs and where most of the effort — typically 50 percent — is expended. Because of this, it is essential that the team enter Construction with a stable architecture (it's not possible to work in parallel without one) and therefore important that the team be honest about the stability of the architecture when exiting Elaboration. Because of the staffing increase and the degree of parallel work required in this phase, entering Construction with an unstable architecture is asking for trouble.

Construction culminates in the **Initial Operational Capability (IOC) Milestone**, which involves assessing the product's suitability for delivery to the end users. The IOC milestone is successfully achieved when the product is complete enough to deliver to users and ready
for evaluation by selected stakeholders from the intended user community in a "beta" deployment. Milestone assessment is based on the product’s maturity, quality, stability, and completeness, as well as the user community’s readiness to accept its delivery. In other words, does the product meet users’ needs sufficiently for some customers to take early delivery?

**Transition Phase Goals**
The Transition phase has these goals:

- Bring roll-out risks under control.
- Deliver the solution to its end users.
- Achieve user self-sufficiency.

Transition starts with the "beta" deployment and concludes with final delivery of the solution to the customer or their support organization. This phase focuses on fixing remaining defects, training users, and, in many systems, converting data from older systems (or older versions of the same system) and running in a parallel testing mode for some period to ensure that the system is ready for final deployment.

Transition culminates in the Product Release (PR) Milestone, which marks project completion. The PR milestone is successfully achieved when the product/solution has been successfully deployed and the maintenance and support responsibilities have been handed over to another project or department. Milestone assessment is based on user and stakeholder satisfaction: Is everybody happy with the result from the project?

**Correcting Misconceptions About RUP Phases and Milestones**
The RUP lifecycle is often misinterpreted by people who are new to iterative development practices, especially those who have previously used a waterfall lifecycle approach in which the phases were aligned to individual process disciplines (e.g., requirements, design, code and unit test, system test, etc.). In this section we will explain what phases and milestones mean in the RUP context.

**Phases and Risks**
A common error is to think that RUP phases are aligned to either the execution of one or two disciplines or the completion of one or two deliverables. Although these associations may have some validity, they are often misleading and result in unproductive or misguided efforts. In fact, it is more correct and often easier to remember the risks associated with the phase, which are shown in Table 3.

**Table 3: Incorrect and Correct Interpretations for RUP Phases**

<table>
<thead>
<tr>
<th>Phase</th>
<th>Incorrect Interpretation</th>
<th>Correct Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td>High-level requirements</td>
<td>Business risks</td>
</tr>
<tr>
<td>Elaboration</td>
<td>Detailed requirements and/or design</td>
<td>Architectural/technical risks</td>
</tr>
<tr>
<td>Construction</td>
<td>Implementation and development; team testing</td>
<td>Logistical risks (the risk of not getting all the work done)</td>
</tr>
<tr>
<td>Transition</td>
<td>Acceptance testing</td>
<td>Solution roll-out (delivery) risks</td>
</tr>
</tbody>
</table>

**Milestones and Achievements**
Newcomers to RUP often misinterpret milestones in a similar way. It is accurate, and often easier, to remember the achievement associated with the milestone, as shown in Table 4.

**Table 4: Incorrect and Correct Interpretations of RUP Milestones**

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Incorrect Interpretation</th>
<th>Correct Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lifecycle Objectives (LCO)</td>
<td>Planning completed</td>
<td>Project viability has been confirmed by stakeholders.</td>
</tr>
<tr>
<td>Lifecycle Architecture (LCA)</td>
<td>Specification completed</td>
<td>The selected approach has been proved via developer testing.</td>
</tr>
<tr>
<td>Initial Operational Capability (IOC)</td>
<td>Coding completed</td>
<td>A usable solution is available.</td>
</tr>
<tr>
<td>Product Release (PR)</td>
<td>Product available/deployed</td>
<td>The project is complete.</td>
</tr>
</tbody>
</table>
Taking a Risk (Phase) and Objective (Milestone) View of Process

Now that you know how to correctly interpret and focus on the risks associated with the phases, and the achievements associated with the phase milestones, it is easier to construct and validate iterative project plans. You can select the appropriate activities to perform and artifacts to produce, to demonstrate that you have mitigated the appropriate categories of risk and achieved the milestone for each phase.

Our high-level risk- and objective-based view of the process is summarized in Table 5.

Table 5: A High-Level Overview of the RUP Project Lifecycle11

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Risk Focus:</strong></td>
<td><strong>Risk Focus:</strong></td>
<td><strong>Risk Focus:</strong></td>
<td><strong>Risk Focus:</strong></td>
</tr>
<tr>
<td>Business</td>
<td>Architectural</td>
<td>Logistical</td>
<td>Roll-out</td>
</tr>
<tr>
<td><strong>Questions:</strong></td>
<td><strong>Questions:</strong></td>
<td><strong>Questions:</strong></td>
<td><strong>Questions:</strong></td>
</tr>
<tr>
<td>Are we building the right thing for the customer?</td>
<td>Do we know what we are building?</td>
<td>Are we getting it done?</td>
<td>Is it acceptable?</td>
</tr>
<tr>
<td>Is the solution feasible?</td>
<td>Do we know how we will build it?</td>
<td>Will it be done on time?</td>
<td>Is it being used?</td>
</tr>
<tr>
<td>How much will it cost?</td>
<td>Do we agree on what it is?</td>
<td>Is it good enough?</td>
<td>Have we finished?</td>
</tr>
<tr>
<td>How will the technical risks be mitigated?</td>
<td>How much will it cost?</td>
<td>Do our assumptions and earlier decisions hold?</td>
<td></td>
</tr>
<tr>
<td>Can we prove that we can mitigate the technical risks?</td>
<td></td>
<td>Are the users ready?</td>
<td></td>
</tr>
</tbody>
</table>

**Key Artifacts:**
- Vision
- Business case
- Risks
- Overall project plan
- List of critical use cases

**LCO: Viability Agreed**
- Use-case model survey
- Detailed descriptions for architecturally significant use-case flows and supplementary specifications
- Architectural description
- Architectural prototypes
- Executed tests of the architecture

**LCA: Selected Approach Proven**
- Use-case descriptions
- Supplementary specifications
- Designs
- Code
- Tests
- Test results
- Training materials
- User documentation

**TOC: Usable Solution Available**

**PR: Project Completed**
- Installers, including data conversion
- Customer surveys
- Defects and their resolutions

**Outcome:**
- Agreement to fund the project
- A stable, proven, executable architecture
- A useful, tested, deployable, and documented solution
- The solution is in “actual use”

Forces That Drive an Iterative Process

Although people often talk about use-case driven development, it is naive to think that use cases are the only force working on the project and shaping project plans. We should always remember that it is really the stakeholders who drive the project. They are the primary source of requirements, constraints, and risks for the project. They supply the funding and audience for the project and decide at the end of each phase whether the project will proceed.

Use cases are a way to express project requirements, and internally they help drive the project by unifying the project team and determining many of the development activities. They can also facilitate understanding of stakeholder needs and working with stakeholders to define the system scope and requirements. Clearly, use cases have an essential role to play in managing and controlling an iterative project, but they are not the only factor to consider. In the next subsections we will look at the other forces that shape the project, how these relate to the phases and milestones of the iterative lifecycle, and the role that use cases can play in controlling these forces by driving activities to achieve the project’s objectives.
Stakeholder in Three Domains

In our book *Use Case Modeling*, we define a stakeholder as:

...an individual who is materially affected by the outcome of the system or the project(s) producing the system.\(^\text{12}\)

We have also found it useful to classify stakeholders by the domain of the system that affects them most:

- **The Problem.** People in this group are affected by the problem or problems that the project intends to solve. They are typically the primary source of requirements; they confirm that the problem has been solved when they accept the product.
- **The Solution.** People in this group are affected by the solution because they have to support its operation or adapt their jobs to it in some way. These people may or may not benefit directly from the solution.
- **The Project.** People in this group are responsible for delivering the solution to other stakeholder groups.

*It is the risks, constraints, and objectives in these three domains that shape the overall project plan:* the number, length, and style of the iterations; the disciplines that should be applied; the artifacts and techniques that are applicable; and the things that need to be done to prove to the stakeholders that a phase has been completed and its milestone achieved.

*As nearly every project has stakeholders in each of these three domains,* nearly every project has to handle risks related to each of them. Even more important, every project must satisfy stakeholders that it is making sufficient progress to address these risks. Considering the project from the perspective of these three domains provides us with a way to simplify planning and management of the project's iterations.

With these domains in mind, we can approach the software development lifecycle in one of two ways: by focusing on the achievements in each domain, or by focusing on the activities. We'll discuss these two possible approaches below.

**Focusing on Achievement.** Table 6 analyzes the iterative project lifecycle from each of the three domain perspectives, presenting an overview of the achievements required in each domain to successfully complete the project milestones.

**Table 6: An Achievement-Based Overview of the RUP Project Lifecycle**
As we look more deeply into overall planning of the project, and then for each of the phases, we will use this framework to present simple, risk-based planning patterns that you can use to form your own project plans. These patterns, which grow out of the domain perspectives, make the RUP easier to understand by focusing on what we are trying to achieve and not on the details of the RUP’s nine disciplines or the Extended Unified Process’s eleven disciplines. This framework can serve as an organizing principle to explore the management of an iterative project, touching indirectly on the RUP disciplines.

**Focusing on Activities.** The three stakeholder perspectives — problem, solution, and project — are also useful for examining the role use cases can play in driving iterative system development. Table 7 provides an overview of the key activities to be undertaken for each area and each phase. It also illustrates the impact that use cases can have on the planning and execution of a software development project.

Of the 60 key activities shown in the table:

- 18 are directly driven by the use cases. These are shown in **bold**.
- 22 benefit from the use of use cases on the project. These are shown in *italics*.
- 20 are not significantly impacted by the use of use cases. These are shown in a regular font.

### Table 7: A High-Level, Activity-Based Overview of the RUP Project Lifecycle

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Problem</em></td>
<td><em>Vision</em></td>
<td><em>Requirements</em></td>
<td><em>Product</em></td>
</tr>
<tr>
<td>understood</td>
<td>agreed upon</td>
<td>correct</td>
<td>accepted</td>
</tr>
<tr>
<td><em>Value and</em></td>
<td><em>Requirements</em></td>
<td><em>Ready to deploy</em></td>
<td>complete</td>
</tr>
<tr>
<td>scope of</td>
<td>stable</td>
<td><em>Acceptance</em></td>
<td><em>Product</em></td>
</tr>
<tr>
<td>solution*</td>
<td><em>Success evaluation</em></td>
<td>criteria agreed upon</td>
<td>deployed</td>
</tr>
<tr>
<td><em>Alignment</em></td>
<td><em>Requirements</em></td>
<td><em>User</em></td>
<td><em>Users</em></td>
</tr>
<tr>
<td>with business'</td>
<td>stable</td>
<td><em>documentation</em></td>
<td>self-sufficient</td>
</tr>
<tr>
<td>goals<strong>verified</strong></td>
<td></td>
<td>available</td>
<td></td>
</tr>
<tr>
<td><em>Critical</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>requirements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>identified*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Solution</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Technical</em></td>
<td><em>Architecture</em></td>
<td><em>Implementation</em></td>
<td><em>Product</em></td>
</tr>
<tr>
<td>feasibility</td>
<td><em>Proven</em></td>
<td>stable</td>
<td>complete</td>
</tr>
<tr>
<td>assessed</td>
<td><em>Executable</em></td>
<td><em>Useful, quality</em></td>
<td>code complete</td>
</tr>
<tr>
<td><em>Solution</em></td>
<td><em>Architectural</em></td>
<td><em>Product</em></td>
<td>maintenance and</td>
</tr>
<tr>
<td>approach*</td>
<td><em>Baseline</em></td>
<td><em>Available</em></td>
<td>support</td>
</tr>
<tr>
<td>agreed upon</td>
<td></td>
<td></td>
<td>responsibilities</td>
</tr>
<tr>
<td><strong>Project</strong></td>
<td><em>Critical</em></td>
<td><em>Objective</em></td>
<td>handled over</td>
</tr>
<tr>
<td><em>Objectives</em></td>
<td><em>Components</em></td>
<td><em>Quality</em></td>
<td></td>
</tr>
<tr>
<td>agreed upon</td>
<td>defined</td>
<td><em>Information</em></td>
<td></td>
</tr>
<tr>
<td><strong>Project</strong></td>
<td><em>Build</em></td>
<td><em>Available</em></td>
<td></td>
</tr>
<tr>
<td>constraints</td>
<td><em>Buy/reuse</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Project</em></td>
<td>decision*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>constraints</td>
<td>made</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Low-fidelity</em></td>
<td><em>High-fidelity</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lifecycle*</td>
<td><em>Comprehensive</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>plan* agreed upon</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>agreed upon</td>
<td><em>Construction</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Project</strong></td>
<td><em>Planning</em></td>
<td><em>Transition</em></td>
<td></td>
</tr>
<tr>
<td>in place</td>
<td><em>in</em></td>
<td><em>Plans</em></td>
<td></td>
</tr>
<tr>
<td><strong>Project</strong></td>
<td><em>Project</em></td>
<td><em>in</em></td>
<td></td>
</tr>
<tr>
<td><em>Control</em></td>
<td><em>Under</em></td>
<td><em>Under</em></td>
<td></td>
</tr>
<tr>
<td><em>Impact</em></td>
<td><em>Control</em></td>
<td><em>Under</em></td>
<td></td>
</tr>
<tr>
<td>of</td>
<td></td>
<td><em>Under</em></td>
<td></td>
</tr>
<tr>
<td>outstanding</td>
<td></td>
<td><em>Under</em></td>
<td></td>
</tr>
<tr>
<td>changes*</td>
<td></td>
<td><em>Under</em></td>
<td></td>
</tr>
<tr>
<td>understood</td>
<td></td>
<td><em>Under</em></td>
<td></td>
</tr>
<tr>
<td><strong>Project</strong></td>
<td><em>Elaboration</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Planning</em></td>
<td><em>Elaboration</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>plans* in place</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Project</strong></td>
<td><em>Elaboration</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>Planning</em></td>
<td><em>Elaboration</em></td>
<td></td>
<td></td>
</tr>
<tr>
<td>plans* in place</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Stable, but not “frozen”.*
By using use cases in conjunction with an iterative lifecycle approach to shape and influence the project plans, the project manager can ensure that development is driven by the requirements, the correct risks are being addressed, the project is converging on the correct solution, and the benefits promised by the iterative approach are realized.

Summary
This article introduces the basic concepts of iterative development, a time-boxed approach to delivering software-based projects that uses systematic reduction of risk as its organizing principal. We looked at iterations and phases and their goals and milestones to understand both risk- and achievement-based views of the RUP iterative process. We introduced three perspectives based on the domains that most concern project stakeholders: problem, solution, and project. These three perspectives provide us with a framework for forming goals for the project and its phases and iterations, to ensure that the needs of stakeholders are met. We can also apply this basic framework to many different kinds of projects: software development projects; process improvement projects; course development; and writing a book, for example. Once you understand the basic framework, it is easy to choose the right set of activities from the RUP disciplines, and even to adapt the RUP to new circumstances we have not encountered or imagined.

Additional Reading
- User experience storyboards: Building better UIs with RUP, UML, and use cases by Jim Heumann
- Adopting use cases
  Part I: Understanding types of use cases and artifacts
References


Notes


2. Much of this information was initially presented more than a decade ago by Ivar Jacobson et al., in *Object-Oriented Software Engineering* (ACM Press, 1992)


4. Iterate (v.): to say or do again; repeat. Iteration (n.) Iterative (adj.).


6. Increment (n.): an increase or addition, especially one of a series.

7. A *time box* is a period of time in which project work is done. Time boxes (in our case iterations) are clearly bounded; at the end of the time box we call a halt to work and evaluate our progress, rather than continuing to work until we are done. Time-boxing establishes a sense of urgency and an awareness of schedule constraints.


9. I.e., establishing an initial architecture, as a basis for subsequent change management.

10. Typically in the form of a beta release.

11. There is a lot of information presented here, and we will use this framework quite a lot when discussing what happens when, and why.
This definition is based on the RUP, which defines a stakeholder as anyone who is materially affected by the project's outcome, and on Dean Leffingwell and Don Widrig's *Managing Software Requirements: A Unified Approach* (Addison-Wesley, 2000), which defines a stakeholder as an individual who is materially affected by the outcome of the system. This new definition recognizes that the stakeholder community comprises both the individuals directly affected by the system and those that are indirectly affected by the system because of their involvement in the project.

The Extended Unified Process is an extension of the Rational Unified Process proposed by Scott W. Ambler and Larry L. Constantine in their series of books on the Unified Process Phases. The Extended Unified Process adds two disciplines (Infrastructure Management and Operations and Support) as well as a phase (Production) to the RUP framework.

About the authors

As Technical Team Lead for Rational UK's Process and Project Management Team, Ian Spence specializes in the adoption, implementation, and configuration of the Rational Unified Process (RUP). In addition to working with the process for more than five years, instantiating it in companies both large and small, Ian has made several contributions to the process itself, completing two secondments to the RUP Business Unit. Currently he is working on a Rational Process Plug-In that addresses program management issues. Prior to joining Rational, he spent twelve years working in the financial sector, including eight as an OO specialist and architect. He holds a degree in Control Systems and Computing Science from Sheffield University in the United Kingdom.

Worldwide communities of practice architect Kurt Bittner joined Rational Software ten years ago and has worked in the software industry for more than twenty-one years. Improving the effectiveness of software development organizations, especially through software engineering management and process change, has been his principal focus. A member of the original IBM Rational Unified Process, or RUP, development team, he later managed the RUP development organization as well as other product development organizations inside IBM. In addition to co-authoring *Use Case Modeling* (Addison-Wesley, 2002), he has also contributed to several other books and authored numerous articles.

What do you think of this document?

- Killer! (5)
- Good stuff (4)
- So-so; not bad (3)
- Needs work (2)
- Lame! (1)

Comments?

Submit feedback