What is iterative development? -- Part 1: The developer perspective

Ian Spence  
Kurt Bittner

from The Rational Edge: This first article of a three-part series examines what it means for software development project team members to work iteratively and incrementally. The authors introduce the theory behind the iterative and incremental development approach and consider how members of the development team experience these methods.

"Iterative and incremental development is necessary to converge on an accurate business solution." -- Principle 5, Dynamic Systems Development Methodology

What does it mean for a project to work in an iterative and incremental manner? This series of articles addresses this question. We first define "iterative and incremental development," then we examine this way of working from a number of the most commonly adopted perspectives (project manager, developer, customer, etc.), clarifying and illuminating what it really means to work in an iterative and incremental fashion.

Part 1 of this article focuses on how the development team experiences the iterative lifecycle. Parts 2 and 3, to be published in the next two issues of The Rational Edge, will examine the experiences of the customer and management teams, respectively.

Iterating and the scientific method

There are many activities involved in developing a solution to a problem. We need to understand the problem, gather requirements for a potential solution, translate those requirements into a design, build the solution, and test it. This order is fairly natural, and generally correct. Problems creep in, however, when we try to scale this up -- that is, when we try to gather all requirements, then do all design, then all development, then all testing in a strictly linear fashion.
Instead, we need to work more like scientists. The modern scientific approach is founded on the principle of direct observation: theories are proposed, then experiments are designed and performed to test those theories. From the measured results, we either reject the theory or confirm it.

How does this apply to software development? In a sense, many things on a software development project are theories, or more accurately, assertions that need to be evaluated. The plan itself is composed of many assertions about how long things will take to accomplish. Even the requirements are assertions about the characteristics of a suitable solution. Just because some stakeholders or subject matter experts say a requirement is valid does not mean that they are correct. We need to evaluate even the requirements to determine whether they define the right solution to the problem at hand.

This leads us to adopt a style of software development where the assertions inherent in the plan are repeatedly challenged and evaluated by the design and development of demonstrable versions of the system, each of which is objectively shown to reduce the project risk and build upon the other to form the finished solution.

This style of development is more commonly known as iterative and incremental development, which we define as:

- A style of development that involves the iterative application of a set of activities to evaluate a set of assertions, resolve a set of risks, accomplish a set of development objectives, and incrementally produce and refine an effective solution.
- It is iterative in that it involves the successive refinement of the understanding of the problem, the solution's definition, and the solution's implementation by the repetitive application of the core development activities.\(^2\)
- It is incremental in that each pass through the iterative cycle grows the understanding of the problem and the capability offered by the solution.
- Several or more applications of the iterative cycle are sequentially arranged to compose a project.

Sadly, development can be iterative without being incremental. For example, the activities can be applied over and over again in an iterative fashion without growing the understanding of the problem or the extent of the solution, in effect leaving the project where it was before the iteration started.

It can also be incremental without being truly iterative. For example, the development of a large solution can be broken up into a number of increments without the repetitive application of the core development activities.

To be truly effective the development must be both iterative and incremental. The need for iterative and incremental development arises out of the need to predictably deliver results in an uncertain world. Since we cannot wish the uncertainty away, we need a technique to master it. Iterative and incremental development provides us with a technique that enables us to master this uncertainty, or at least to systematically bring it sufficiently under control to achieve our desired results.
The iterative experience

If you survey the literature for methodologies that describe the various forms of iterative software development, you will find that there are a number of different interpretations of what it means to iterate and what makes a suitable increment. These tend to reflect the role that the authors played on their projects and the particular aspects of iterative development that were considered to be of most importance within their project teams, organizations, and peer groups.

Underlying these various interpretations is the varied experience of iterating, and working on an iterative project, which appears quite different to different team members involved in the project. Let's consider how adoption of iterative development impacts the most common roles involved in a software development project.

To organize the material, we have found it helpful to consider the roles grouped into three teams:

• **The Core Development Team.** These are the people focused on formulating and developing the solutions identified by the requirements, including the application of the core development disciplines of Architecture, Analysis, Design, Implementation, and Test to develop high-quality components and solutions.

• **The Customer Team.** These people are focused on defining the problems to be solved and the things that will be built (including changes to business processes) to solve these problems. They must ensure that any solutions produced provide sufficient benefit to the commissioning organization.

• **The Management Team.** These are the people focused on making sure that the customer, business, and development goals are aligned, the right problem is being solved, the right solutions to these problems are being built, and the development is undertaken in an effective, efficient, and controlled manner.

In the rest of this article, and the two that follow, we will take a closer look at iterative development from each of the three perspectives outlined above, examining how iterating helps the project achieve all of its goals and how it affects all of the people involved in the project.

Iterating from the core development team perspective

Let's consider project dynamics from the perspective of the core development team. This team is responsible for applying the development disciplines of Architecture, Analysis, Design, Implementation, Unit Testing, and Integration Testing to produce releases of the system that fulfill the customer's requirements. The requirements are considered to be the responsibility of the customer team, even though there may be customer representatives, or business analysts, assigned to work directly and permanently with the development team. (We will look at iterating from the customer team's perspective in the next article.)

The developers' perspective

The most common perspective taken on iterative and incremental development is that of the individual developer working on a small development team. After all, this is the perspective of the technical community that has produced the majority of the literature about iterative and incremental development. They have also provided most of the impetus for the adoption of these practices.
One of us recently attended a talk at a software developer symposium. The subject was iterative development using a specific interactive development environment.

It soon became apparent that the presenter considered an iteration to be the iterative application of the "write test, write code, test code, fix code" cycle, applied over and over again, until a piece of completed, unit tested code is available for inclusion into a release of some sort. And while it is true that these "development" activities embody many of the best practices recommended by the proponents of iterative development (such as test first, test often, etc.), the way that their application was described would result in a very tight iteration, the duration of which could often be measured in minutes.

This view of an iteration as a single developer's cycle of "write test, write code, test code, fix code" seems to contradict the classic management view of an iteration as a time-boxed mini-project undertaken by a team of developers. It represents a very egotistical view of iteration focusing solely on the individual's contribution rather than the team's. This also contrasts with the broader team leader's perspective, which we'll discuss a little later. For now, let's focus on the individual developer's perspective.

Just as the symposium presenter described it, the tight, iterative loop illustrates the natural way in which developers work on iterative and incremental projects. Individual developers follow their own rhythm so that the traditional phases of analysis, design, and implementation start to blur and their daily activities come under their own direction. At any one time, myriad tactical decisions are being made by the individual developers as part of the fabrication and adaptation of the system's software. Individual developer activities tend to be synchronized around shared events, such as the daily build and system integration points. Group activities tend to be focused on architectural work, interface definitions, and the re-factoring of components.

Once developers understand the limit of their responsibilities and their current objectives, their attitude is, typically, to roll-up their sleeves and do whatever it takes to meet their obligations and to deliver one or more working components to be integrated into some form of release. Each individual developer (or pair of developers if practicing pair programming) will develop an individual way of working within the framework laid down by the processes adopted by the project.

**Progress and the product backlog**

The individual developer is rarely interested in non-technical things, such as benefits realization, return on investment, or risk management. They achieve closure by regularly building real, executable things. Developers work by selecting sets of requirements and change requests to be analyzed, designed, implemented, built, unit tested, and then included into a release of the product. The overlapping nature of the developer's application of the Analysis, Design, and Implementation disciplines is shown in Figure 1-1. The time and scale of the developers' iterative process can be measured in hours, days, or weeks, depending upon the scale and complexity of the endeavor undertaken.
Developers will continue to work in this way, selecting and implementing small numbers of requirements or change requests from their list of outstanding tasks, until they have completed all of the items that have been assigned to them.

From the developer's perspective, the incremental part of the iterative and incremental development is manifested in the increasing completeness and functional richness of the releases produced. This is contrasted with the, hopefully, ever smaller number of requirements and change requests waiting to be implemented. This list of outstanding requirements and change requests is often referred to as the product backlog,\(^4\) as it represents the backlog of work waiting to be done. Each day the developer takes more items from the product backlog, each day the build contains more implemented items, and each day some new work may be added to the product backlog. This daily, incremental progress through the work in the product backlog is shown in Figure 1-2.

Figure 1-2: Incrementally addressing a product backlog

This developer-centric approach works well for small teams of developers who work closely together on a set of loosely interacting components, or for larger communities of developers who work on a common code base, such as open source projects. To scale up to larger projects of...
closely interoperating components -- typical in larger development efforts -- additional perspectives are needed.

The development team leader's perspective

In contrast to the individual developer, the leader of the development team is concerned with the coordination and optimization of the work of several developers, each of whom is changing or implementing a number of components that must be integrated into a release that fulfills the team's shared objectives.

From the development team leader's perspective, an iteration is not just a cycle through the core development activities of analysis, design, and implementation; an iteration is typically a time-boxed mini-project that results in the production of a significant new release of the software.

Figure 1-3 illustrates how the team harnesses the work of a set of individual developers to produce an integrated release that is then used to generate feedback which shapes the change requests and requirements to be actioned in the next iteration and included in the next release.

Figure 1-3: Iteration from the development team leader's perspective. (Adapted from *Agile and Iterative Development, A Manager's Guide*, by Craig Larman, Addison-Wesley, 2004.)

The goal of an iteration is a release that integrates the work of the whole development team into a stable, integrated, and tested release of the system. For most of the iterations these releases are internal releases -- baselines created primarily for the benefit of the development team and used to provide closure to the iteration and measure the progress being made. These are "internal releases" not actually deployed to the users.

Typically, there are three or more iterations leading up to the public release of the software. Each iteration in the sequence sees the production of a release of the system, starting with a small, partial implementation of the system in the first iteration that then grows incrementally with the implementation of more requirements, iteration by iteration, until the final customer release is produced. Figure 1-3 illustrates this growth with the growing size of the circles representing the releases produced.
Connecting the developer and team leader perspectives

To many developers, iterative and incremental development is the progression through an almost infinite series of iterations, each of which causes the system to grow larger and larger until the customers are happy with it or decide to stop investing in its further development. From this perspective, as long as work is getting done and changes are being made, the project is making progress.

From the team leader's perspective, the work of several developers must combine and integrate, or work together successfully, in order for progress to be made. Figure 1-4 illustrates this holistic, development-team view of progress across a number of iterations.

![Bar chart showing development team progress across iterations](image)

Figure 1-4: Each iteration produces a more complete release than the previous iteration.

In Figure 1-4, progress is measured by how complete the integrated, working release produced by the development team is, and not by the completeness of the specification or other documentation. The completeness of the application is calculated from the number of requirements that the integrated application can be shown to implement. In each iteration, the focus is on the production of a working release and not some set of project documents or un-integrated components.

It is essential that the team's progress is considered as a whole, via the testing of the integrated release. We have witnessed a number of projects in which prodigious numbers of components were created and tested by developers but could not be integrated into even a partially complete product. In those cases the view of the project illustrated in Figure 1-4 never occurred, because there was no objective evidence that any progress was being made on the completed project.

The need for planning

In order for the work of more than a few developers to be integrated, some amount of planning must be done. First, work must be allocated to developers in a way that minimizes the
interdependencies between them, so that they can work with some degree of independence even though they are in fact interdependent on one another.

Second, they must have a way to agree on their interdependencies; for example, they must be able to agree on names of components, function calls, and parameters. As the interdependencies become larger in number, the team eventually reaches a point at which informal interpersonal communication may need to be augmented with a record of the agreements made between the developers.

As these complexities increase, and as the effort to efficiently coordinate the work of the team members becomes more complex, the team leader will need a way to understand what will get done when and in what order. This gives rise to the need for more formal planning, which in turn drives the need for estimation. It is the interdependencies between components owned by different developers that initially creates the need for more formal planning, as the schedule can no longer be determined by simply either taking the expected development time for the most complex component (the case where all components are independent) or summing the development time for all the components (the case where all the components are so interdependent they can only be worked on one at a time). The dependencies between components must be understood to construct an effective schedule.

Iterative development is sometimes said to operate on such short time scales that there is no need for planning and estimating. This argument is specious; it is a little like saying that since we only take one step at a time there is no need to plan our route since one can make continuous decisions about where to turn next. As long as you don't care where you end up, this strategy works just fine, but the people paying for the project most certainly do have goals that they need the project to reach.

Planning and estimating is basically a mechanism for first determining that the project is worth doing and later as an approach for keeping the project on track. Few of us would hire someone to repair their home without knowing how much it will cost and how long it will take at least with some degree of certainty. Software development projects are no different: the people funding the project need to know how much it will cost and when it will be done. That requires both planning and estimates. We will return to these topics when we look at the customer and management team perspectives in the next two articles.

**Conclusion**

To understand how to achieve the benefits promised by iterative and incremental development, it is important to understand how it will affect all the people involved in the project. In this article, we have looked at how iterative development affects the people who are producing the software, the members of the core development team, but this is only one set of people involved in the project. If we want our iterative projects to be truly effective we must also understand how the adoption of iterative and incremental development processes will affect those other, equally important, areas of the project: the customers and managers.
We will examine these two perspectives in the next two articles, where we will look at how the customer and management teams are affected by adopting an iterative and incremental way of working.

See you next month.

Notes

1 For more on the Dynamic Systems Development Methodology, or DSDM, see the GamCom Solutions Web page

2 The core disciplines are Project Management, Requirements, Analysis, Design, Implementation, Test, Deployment, Configuration Management, and Change Management.

3 Throughout this series we will use the term "implementation" to refer to the development discipline that encompasses the coding and unit testing of a component. This is the name used for the discipline within the Rational Unified Process.

4 The concept of a prioritized product backlog was codified by, and is central to, the Scrum iterative management method. See Agile Software Development with Scrum 1st edition by Ken Schwaber and Mike Beedle, Prentice Hall; (October 15, 2001).