11

Getting Started with Iterative Project Management

"Desirable ends do not come of themselves. Men must conceive them, believe in them, further them, and execute them."
—Carl Van Doren

This book has presented an approach to improving business results through the systematic application of the principles of iterative development, but the principles do not apply themselves. In this final chapter we present practical advice on adopting these practices within your projects and your organization. We realize that you come to the subject of iterative project management with different goals and different constraints on how much change you can introduce. You might be someone who is

- Completely new to managing iterative development projects and about to start on your very first iterative project
- Already engaged in managing iterative projects and struggling to put the theory into practice
- Already managing iterative development projects and looking to improve and validate your approach

- Engaged in rolling out or supporting the rollout of iterative development techniques to your department or organization

To support all these views and more, this chapter looks at how to you can apply what you have learned to

- Implement your first project
- Build on your current success
- Help change your organization

We discuss how you can start to apply the techniques immediately to the projects that you manage and how you can build upon this success to progressively expand your use of iterative software development practices.

Embarking on Your First Iterative Project

Embarking on your first iterative project probably fills you with some uneasiness, and you might have some doubts about your project taking on what you might perceive to be additional risk. In this section we help you take that first step.

Why Iterate?

As we have noted a number of times in prior chapters, iterative development is undertaken to eliminate project risks early in the project, before they can have a chance to sink the project. By tackling risk explicitly, your project will be more likely to succeed. That alone should provide fairly powerful motivation. Other reasons for adopting an iterative approach include

- To achieve higher quality
- To achieve faster results

- To achieve results more reliably, or sometimes just to achieve results at all
- To reduce staff frustration and turnover
- To achieve greater flexibility and business agility
- To reduce costs

Understanding these and any other goals is important so that you can factor them into the actions that you and your team will take.

The impetus for change can come from any number of sources: from senior management wanting to achieve better business results, from senior technical leaders wanting to resolve recurrent project problems, or from project teams that want to improve themselves. Whatever the source, the stimulus for change tends to gestate until a sponsor picks it up.

There also needs to be a sense of urgency about the need to change; some crisis or significant opportunity is required to get the project team members to be interested in the extra work of learning how to do something new. Change is initiated for a variety of reasons. The reasons listed here are the major themes that lie behind most change initiatives. Often the reasons for the change are not well articulated or even well understood.

Unless there is a sense of urgency, the stimulus for change will never become great enough to overcome resistance. People become comfortable with the status quo, and it is difficult to get them to change unless they feel that they will personally benefit from the change. They need to believe that unless they change the way they work, bad things will happen, or in the case of opportunities, good things will fail to happen.

This is not to say that threatening or frightening people is an effective motivational technique (it is not). It merely observes that people need to feel that if they don't change, they will fail. In the early stages of a change, only a few people might feel this need to change. As long as the change is small enough that it only affects a small group, this is sufficient to move ahead.

Potential Barriers to the Adoption of Iterative Practices

Before you start, you need to understand the potential barriers to the change so that you can make the right choices of timing, project, and approach. Some of the more important questions to ask include in the following:

- **How supportive is senior management of the change?** The measurements and milestones they establish can easily derail an iterative approach, as is the case when they ask even seemingly innocent questions such as "When will the design be completed?" or "When will requirements be signed-off?"

- **What is the scope of your authority to make changes?** How much of the development lifecycle are you responsible for? For example, the requirements might have already been specified in a format and to a level of detail that would make it more difficult to adopt an iterative approach.

- **What are the team's feelings about the changes?** How enthusiastic is the team about iterating? To achieve the transition to iterative development, you will need the support of the team, especially the other members of the leadership team.

- **What else does the team have to do?** How many other projects and initiatives is the team involved in? If the team is not focused on and dedicated to the project, the transition to iterative practices will probably be slower and take more time and energy to complete.

- **What capability do you need to improve?** It is important to understand the capability of the team and how well the current capability supports the proposed iterative approach. For example, is there any testing capability in the team? Testing will be needed from the first iteration, which is often a problem in companies organized around the phases of a waterfall

process where the expectation is that testers will only be needed late in the project lifecycle.

- **What work has already been done?** Few projects start from scratch. You need to know what products and artifacts have already been produced.

- **Where are you in the project lifecycle?** It is important to understand the state of the project. Changes are easier to make during an evolution's Inception and Elaboration phases than they are during its Construction and Transition phases.[1]

Selecting the right project and the right team members for the initial effort is important. Table 11-1 presents some of the key characteristics to look for in your first iterative project.

Table 11-1 *The Characteristics of a Good First Iterative Project*

Characteristic	Description	Reason
Attitude	Iterative development needs a team that wants to iterate (or at least to try new approaches).	"Unbelievers" will revert to their old ways of working, masking this by using "iterative" terms.
Project Size	The project needs to big enough to have at least four iterations but small enough that it will deliver results in four to six months. You want to choose projects that can demonstrate rapid success in dealing with real problems.	It will take a few iterations for the benefits to be accepted by the team. The team will need time to get used to the new ways of working.
Team Composition	Iterative development requires a full development team to be in place.	The team might be small but needs to cover all the software development disciplines all the time. Testers are needed early and throughout the project, not just at the end.

[1] You can still evaluate a project using the Unified Process lifecycle even if the project is not being run using the Unified Process and even if it is not being run iteratively. The questions in the phase checklists can be applied to any project, and the answers will provide valuable insight into the project's state and outstanding risks.

Table 11-1 *The Characteristics of a Good First Iterative Project (continued)*

Characteristic	Description	Reason
Technical Leadership	You need the support of people who can accurately gauge technical risk and help you to use this assessment to drive the definition and mitigation of technical risks.	The selection of appropriate risk reduction strategies requires a high level of technical knowledge about the proposed solution. The management team relies on the architecture team to successfully guide the project through the Elaboration phase.
Business Criticality	The project needs to be business-critical enough to get the attention and involvement of stakeholders throughout the project.	Stakeholder involvement is needed throughout the project if the full benefits of iterative development are to be achieved.

The lack of these characteristics does not, by itself, present an insurmountable problem, but it will slow the pace of the project, in particular extending the first two phases where the major business and technical risks are addressed.

When you start your first iterative project, factor the successful adoption of iterative techniques into the critical success factors for the project and the career development objectives for the team members. This is especially important if circumstances require an investment in training and mentoring the team to enable the transformation to take place.

Conventional wisdom suggests that you should choose low-impact, non-critical projects to be the focus of early change efforts to reduce the change effort's risk and the potential for failure. The problem with this is that although these projects are lower risk, their non-criticality usually means that they are starved for resources and not considered "mainstream" enough to ever be taken seriously as success stories. This dooms the overall change initiative. Low-priority projects are not visible enough to engender the support needed to drive the change forward.

Instead you should

- Look for a *small* number of must-do projects that have organizational focus and strong support

- Look for projects with a smaller set of stakeholders to keep communication simpler
- Look for projects with high internal visibility—ones that would make good success stories
- Choose projects that are short- to medium-term in length
- Staff the projects with the best people available (respected leaders)
- Organize projects to generate short-term wins—divide work into iterations
- Keep the project size small in the first few iterations and then scale up
- Focus on choosing the right projects and committing resources to them

Choosing critical projects sounds risky, but there is no sense in hiding from the fact that only critical projects will get the attention and resources necessary to succeed. The key is choosing projects critical enough to get resources but that can start small and then scale up in a controlled way, not becoming too large before the architecture and technical risks can be brought under control. Scaling up should be targeted for the Construction phase but not before.

Communicating the Goals of Change

To motivate the change in approach, you must be able to clearly and concisely communicate why a different approach is needed. If the business goal is to achieve better responsiveness to changing market needs, the goal for the iterative project effort might be something like *be able to go from idea to released product in nine months (or less!)*. The more precise and measurable the goals, the easier they are to achieve.

It's important for you to do a couple things when you are communicating the reasons for the change:

- **Be concise and articulate, and be able to explain it in a few minutes or less**—The vision needs to be precise and specific, without sounding like empty slogans exhorting people to "do better." It should set specific goals that can be translated into criteria by which people can make decisions.

- **Link problems to outcomes, such as "If we don't do X better, Y will happen"**—Linking problems to outcomes is essential to getting a sense of urgency to solving the problem. Being specific is also important. Generalization is easy but not very compelling. Ultimately you need to set specific goals. Being specific about the problem and its impact sends this message from the beginning. For example, "Our inability to deliver releases within X% of the projected date results in lost opportunities of Y million dollars per year" directly links the problem to its outcome.

You must communicate this vision at every possible opportunity, at every level of the organization. Everyone should understand what is being done and why. Learn to be able to concisely describe the vision in a few minutes and present it as an "elevator pitch"—think of a chance meeting in an elevator with a key decision maker. You have this person's undivided attention for, at most, a few minutes. The vision needs to be crisp, concise, and compelling enough to explain in just these few minutes. Everyone today is busy; they don't have time to carefully read pages and pages of reasoning and justification. Set it out for them, and your message will have a better chance of being internalized. The vision need not be very specific about how the change will be achieved, but it does need to be compelling.

John Kotter[2] observes that failure to communicate the vision is a key factor in failed change efforts. You will probably need to communicate the vision 10 to 100 times more often than you are planning to. People usually focus on the impact and cost of changing; everyone needs to understand the impact of not changing. Only when people feel that their outcomes will suffer if they don't change will the change really take root.

[2] John P. Kotter, *Leading Change* (Harvard Business School Press, 1996).

Determining the Pace of Change

Every person and every project has a limited tolerance for change. People who have been successful with change initiatives in the past are more likely to embrace change, whereas people and projects that have a mixed or poor record of success with change will be more resistant to change. Trying to make too many changes too fast will be destabilizing and will make things worse, at least for a period of time. Most people have a threshold to the pace of change that they can sustain—try to push change faster, and the entire effort can stall and fall apart. The ability of the people to deal with change needs to be taken into account when planning change.

If the change is too large, people may lose hope that it will ever pay off and might abandon it. We have personally witnessed this many times, which is why we recommend an iterative approach to introducing change. This seems like simple common sense and no great innovation, but it remains a mystery to us why people so often try to push large changes in a single large initiative.

Expectations must be managed carefully. The tendency for teams to want to do everything immediately needs to be tempered; a sense of "proportion and pace" is crucial. The improvements must be driven fast enough to achieve the desired results as quickly as possible, but not so fast that the team gets confused or disheartened at the lack of progress caused by trying to do too much change at once.

Dealing with Skepticism

You will encounter skepticism and disbelief in the approach that you are proposing. You should be prepared to answer the skeptics because they are likely to become your biggest supporters if you can win them over. Everyone has seen grand new approaches that claim wonderful benefits but fail to produce results. The skeptics will ask, "What will be different this time?" The argument is based on the observation that every project manager starts every project with a new plan and the best of intentions, but the result is always the same: project slips, frustration, and failure. You must be able to answer the question, "Why should we believe you are doing something significant enough to affect the outcome?"

We have encountered this question often enough to have some thoughts on how to answer it. Appropriate responses include the following:

- "We recognize that change is inevitable, and we are taking an approach that recognizes that. We will set goals for our progress, we will measure ourselves against those goals, and we will adapt our approach in light of new information rather than assuming that we had all the answers at the start of the project, which you know is not true."

- "We will focus on the big risks early, while we can still do something about them, and we will take explicit action to reduce those risks. If our first attempts do not work, we will keep at them until we have dealt with the risks."

- "We will focus on delivering the most important things the business *really needs* in an agreed-upon time frame rather than delivering everything they want. This is an important change from how we have done things in the past."

In short, you need to say and show how the iterative approach is pragmatic and deals with the realities of the world. Although real convincing comes only with the proof provided by real progress, most skeptics find the honesty of statements like these to be refreshing, provided that they are backed up with action.

Starting with Just Iterative Development

As discussed in Chapter 1, "What Is Iterative Development?" iterative behavior can be thought of as starting from the activity of developers. For a project to be considered iterative, it is essential that the software be developed iteratively—that is, the core development disciplines of Analysis, Design, and Implementation (which includes developer-driven testing) are applied repeatedly to evolve the software. Fortunately, this is the most natural way for developers to work. As Figure 11-1 shows, when these disciplines are executed in an iterative fashion, additional disciplines can be added around the core development ones to get the whole team iterating in an effective and collaborative manner.

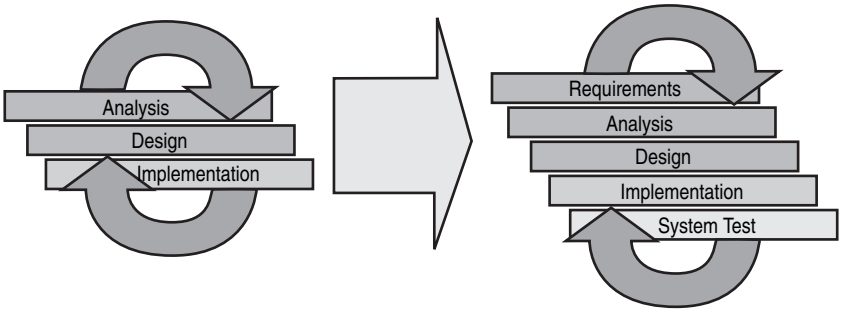

Figure 11-1 *Development (Analysis, Design, Implementation) lies at the core of the iterative approach.*

In some organizations, where waterfall development has become so entrenched that it has influenced the organizational structure and management responsibilities, you might find that

- The development work does not start until the requirements work is complete, and/or
- The other stakeholders and managers in the organization are very suspicious and don't believe that iterative development is suitable for them.

In this case you can still apply and benefit from iterative development and project management techniques by adopting the "requirements pipeline" pattern last seen in Chapter 1 and repeated in Figure 11-2.

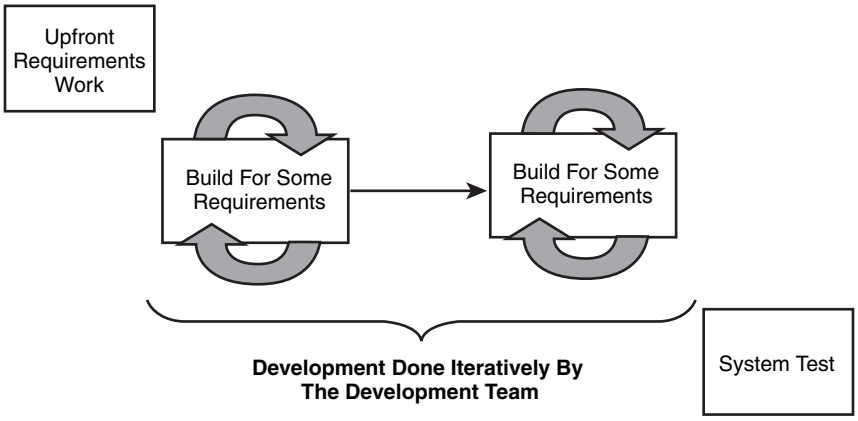

Figure 11-2 *Developing iteratively inside a waterfall requirements model*

This compromise is often adopted in organizations new to iterative development, and it reflects the fact that the desire to iterate commonly originates in the development teams. To prove their iterative development capability, they will start to develop the software iteratively while the requirements and overall system testing are done in the traditional manner. In this case, the job of the development team is made easier if the requirements are captured in a form that enables the identification of sensible chunks of work to be implemented in the iterations.

After the development team has demonstrated the capability to implement sets of requirements in an iterative fashion, you can then expand the scope of the iterations to include the requirements and assessment disciplines. The projects will not be truly iterative in the way we have described throughout the rest of the book until you change the way that the development team works with the business.

Bootstrapping an Iterative Project

Starting your first iterative project is really just like starting any other iterative project. You apply the "bootstrapping" process described in Chapter 7, "Evolution and Phase Planning":

- Start with a small team focused on the business and technical risks
- Challenge the team to start with a four-week iteration
- Adjust the iteration length in response to work and team culture
- Plan to deliver business benefit every three to six months
- Plan releases as well as iterations
- Start with a colocated team
- Address the architectural risks before scaling up

Regardless of how large the project might appear to be, remember the principles of scalable iterative project management presented in Chapter 5, "A Layered Approach to Planning and Managing Iterative Projects," Chapter 6, "Overall Project Planning," and Chapter 10, "A Scalable Approach to Managing Iterative Projects":

- **Start with the assumption that it is small**—If you start big, it will stay big, and generally the larger a project, the more likely it is to fail.

- **Do everything within your power to keep the project small**—If you don't fight to keep things small, they will naturally tend to get large.

- **Layer the plans and keep them succinct and focused**—To be comprehensible, plans must be small. Exploit the layering inherent in iterative projects to avoid unnecessary precision in planning the work.

The first iteration of your first iterative project will be the hardest to plan and manage; chances are you won't know exactly what to do and neither will your team, and this uncertainty is unsettling. It is also likely to be a new experience for the stakeholders as well as the project team, which will only compound the problems. To compensate for this, make sure that the iteration has modest objectives: whatever you do, don't set yourself and the team up to fail by setting ridiculously aggressive objectives.

If you are following the advice presented in this book, then the iteration should be of average length (we recommend four weeks but are prepared to accept four to six weeks as a starting point). It is common for initial iterations to run over their schedule. Allowing this to happen establishes a bad precedent. Keep an eye on progress during the iteration and be ready to scale back on ambitions even in the middle of the iteration to make sure that you will have time to assess results. This usually means moving some scope from the current iteration to the next in order to maintain the iteration time box. When you assess the iteration, you can decide whether to make future iterations longer.

Your first iteration will probably be over-planned. Don't worry; just keep learning and looking for ways to simplify the plans. The main trap to watch out for is overly aggressive planning, especially overestimating the productivity of a new team adopting new practices. If the team becomes demoralized, it will be impossible to get the project back on track, so set reasonable goals early and then ramp up expectations when the team is executing effectively.

Overly aggressive planning has other side effects on the expectations of external stakeholders. These stakeholders tend to focus almost exclusively on whether you did what you said you were going to do, and they will notice if you miss a milestone or if the project consistently fails to meet the commitments you have made to them and the other parts of the organization. The results of overly aggressive planning are illustrated by the following story.

> *We once met a project manager who, having failed to complete the Elaboration phase in the predicted number of iterations, needed to adjust his plans. The reason that the phase could not be completed successfully was that the architectural baseline could not be tested because no hardware was available to run the tests.*
>
> *On inquiring of the supplier when the hardware was likely to be available, the supplier responded that delivery would take at least six weeks, and that could only be achieved because they were such a valued customer and could be promoted to the top of the queue.*
>
> *Learning this did not stop the project manager from re-planning the phase by adding one four-week iteration to the plan. The hardware predictably failed to magically appear earlier than promised, and the project again failed to complete the testing. This repeated failure to achieve the milestone led to a complete loss of trust between the project manager and the steering committee, and caused the manager's removal from the project.*

It will take some time and effort to achieve the full benefits of an iterative approach. Your first evolution is unlikely to accomplish more than successfully producing the same amount of software in a non-iterative fashion, but there will be less rework, the project will be less risky, and the chances of delivering in the predicted timescales will be increased. As a result, it is

important that you do not set unrealistic expectations for what can be achieved. Iterative development is not necessarily *faster* than traditional development, but you will be more certain of delivering the right result in an acceptable amount of time.

Maintaining Momentum

The more you iterate, the better you and your team will get at it. Each iteration results in management, planning, process adoption, and team interactions improving and iteration becoming easier. By struggling through the first few iterations, you and the team learn how to iterate and how to work together in an iterative fashion. Over time, the levels of planning, reporting, designing, and everything else will settle at a natural level suitable for the team and the project. You achieve this by reviewing results at the end of each iteration and adapting your tactics and plans, discarding or amending things that don't add value, and adding techniques as needed to resolve issues.

It will probably take you and your team a number of evolutions to become really proficient at iterating. The key is to keep going and use the evolutionary and iterative nature of the projects to evolve the team's capability alongside the solutions that they are developing. Achieving continuous improvement through iteration is the subject of the next section.

Adopting an Iterative Approach Iteratively

One of the most powerful things about iterative development is that it provides a platform for continuous process improvement. The adaptive nature of the management and development processes means that you can do more with your project's agility than just respond reactively to change—you can also respond proactively to the lessons learned and the trends exhibited by the project to continuously improve the project's working practices and performance.

Understanding Where to Start

Although we hope that by this point in the book you accept that iterative development and iterative project management are useful tools to deliver better results, we recognize that you cannot get there all at once. Your initial iterative projects are unlikely to exhibit all the desirable characteristics described in Chapter 2, "How Do Iterative Projects Function?"—let alone measure them. In our experience it takes the typical project team at least three evolutions to start to exhibit all the desired behavior and establish themselves as experts in iterative development. *Iterative development* covers a broad set of behaviors and cultural values that emerge over time when teams are focused on results. In choosing an approach to phase in improvements, we have found the model depicted in Figure 11-3 useful when thinking about what improvements must come first.

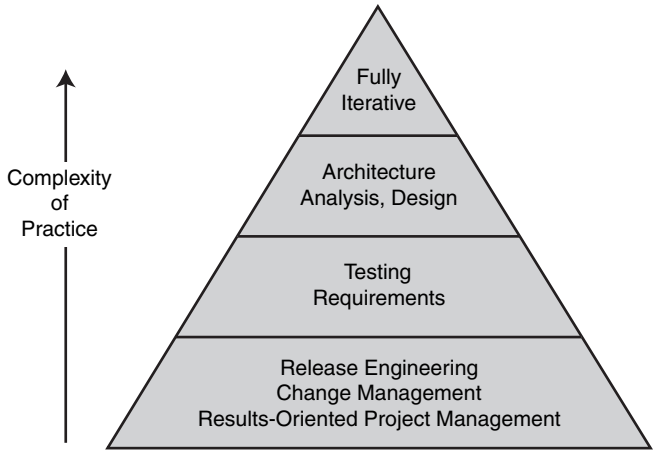

Figure 11-3 *The foundations of software development*

As shown in Figure 11-3, the foundation of iterative development is provided by a shift in project management focus away from creating detailed plans and measuring activities against these plans to results-oriented project management. As we have discussed in earlier chapters, software development is inherently creative and somewhat unpredictable. This means that the precisely appropriate approach is also not entirely predictable. Focusing attention on setting the right goals for each iteration and measuring achievements against those goals is the first and most important step in adopting iterative development.

Introducing effective practices in the areas of release engineering and change management (specifically basic versioning, baselining, and control over the things the project is producing) is essential to supporting this shift to becoming "results oriented." The ability to manage change across iterations (to determine which changes should be addressed in each iteration) and the ability to create executable releases (which requires a reliable build process) are essential to being able to make the shift to delivering objective results.

These basic skills can take you a long way, and in fact they provide most of what you need to perform maintenance and defect fixing across a series of iterations. This covers a great deal of software development activity, and most organizations would gain tremendous benefit if they only did these three things well.

If you are involved in new product development or are making larger and more significant evolutions to the software, you will need to add some additional skills, primarily the ability to understand needs and define requirements (referred to as "Requirements" in Figure 11-3) and requirements-driven testing. If you need to build a completely new solution, you will probably not have the luxury of building upon an existing architecture. You will need to have a more disciplined way of forming the architecture of the system and translating requirements into designs.

As the solutions become more complex, you will need to draw upon all these skills in a fully iterative approach that is able to dynamically respond to new risks and find creative solutions to new problems.

The main reasons for presenting this model are first, to illustrate that most projects can get a lot of value from a basic focus on the fundamentals of results-oriented project management, release engineering, and change management without formalizing the approaches used for requirements, testing, architecture, analysis, or design, and second, to demonstrate that improvements in requirements management, testing, analysis, or architecture must be built on a good foundation of the "lower-order" techniques.

Improving Practices Iteratively

When applying this model to make improvements in a gradual or "progressive" manner, you should not strive to first become "perfect" at the lower levels before moving up to the next level. Instead, we recommend the approach depicted in Figures 11-4 to 11-6.

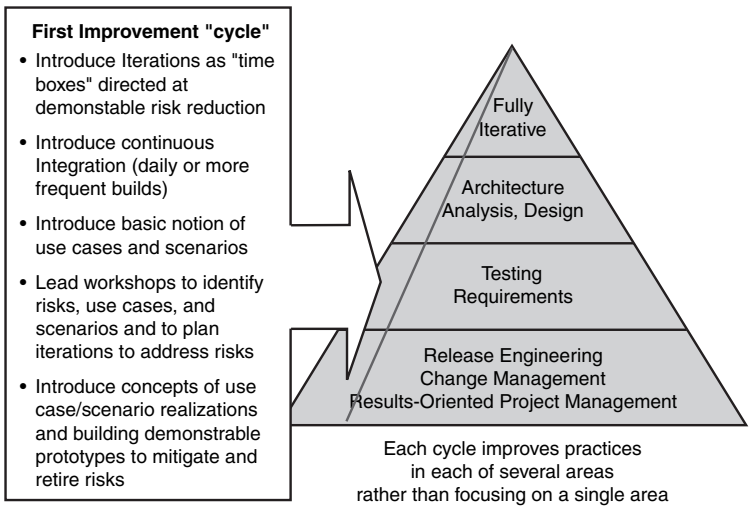

Figure 11-4 *Improvements in the first "cycle"*

We recommend introducing new practices iteratively, in a sense taking a "slice" of a set of the overall practices that are to be introduced and implementing them in a series of iterations. We refer to this "slice" as an "improvement cycle," which could be a single iteration or could be as long as an evolution depending on the scope of the improvements being made. An improvement cycle consists of one or more iterations over which you will introduce some changes and then measure the results. The concept of an "improvement cycle" aligned with a specific set of iterations enables change to be introduced gradually and in a controlled way so that you can make sure that a more basic set of improvements has been successful before you introduce additional change.

As noted in Figure 11-4, early improvement cycles introduce simple concepts such as the notion of an *iteration* as a time box in which specific results are achieved. The shift to a *results-oriented* perspective from an *activity-oriented* one is important, and it represents a big leap for many people, so you don't want to confuse people by introducing lots of other changes at the same time. We have found that the improvements listed in Figure 11-4 tend to be the most important changes to introduce first.

The early improvement cycles are specifically *not* focused on formality or matters of style because this tends to derail success and gets people focused on formality and not results. The key point is to introduce some improvements from each "level" in a lightweight way and only to the

degree that the changes improve results so as not to introduce too much change at once. Figures 11-5 and 11-6 show that as the team becomes comfortable with basic skills, the scope of the improvement effort can expand. Improvement cycles continue as long as the team feels that the improvements are adding value and reducing risk.

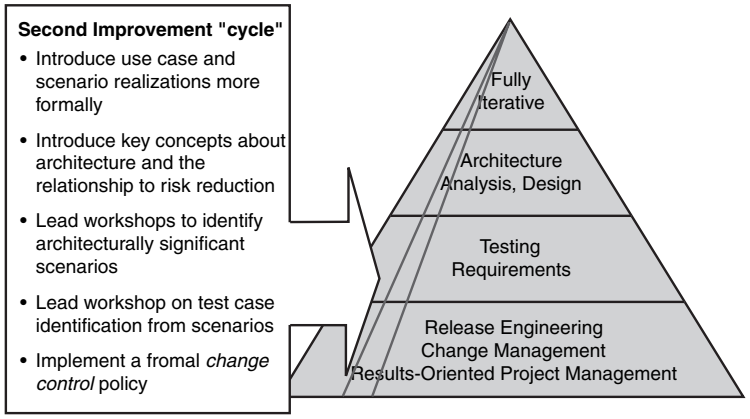

Figure 11-5 *Improvements in subsequent early "cycles"*

Specific improvements are driven by issues identified in iteration assessments to keep them focused on practical project needs. This approach enables the team to make progress and improve results while still getting useful work done. The boxes in the figures indicate the kinds of improvements that are *typical* in early and later improvement cycles.

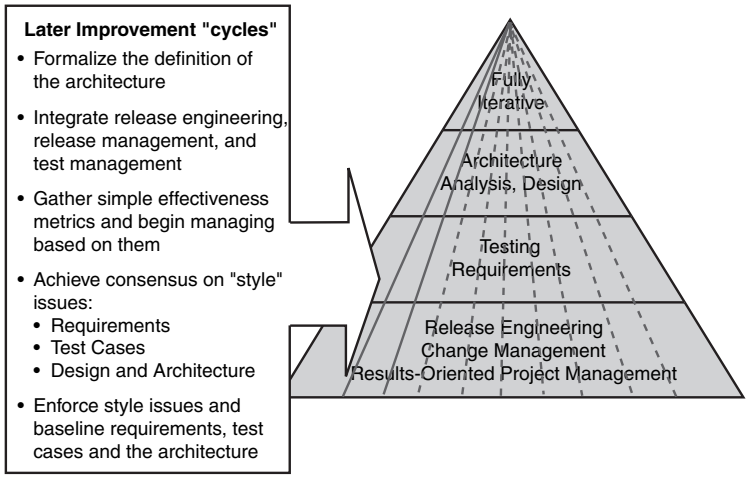

Figure 11-6 *Improvements in later "cycles"*

The essence of this approach is to introduce techniques in each improvement cycle to address the specific needs of the team rather than adopting a set of techniques wholesale. This enables the project team to show results *while* they are improving rather than letting the improvement effort stand in the way of producing results. This overcomes much of the traditional resistance to change.

Learning by Doing

The advantage of this approach is that people learn by doing, and they learn only what they immediately apply. The conventional wisdom (as practiced in much of the industry) is that change is introduced by defining the new processes, selecting supporting tools, and then training everyone on the new processes and tools. As it turns out, this approach actually increases the risk that the change will fail—the opposite of the intended result.

Training is necessary but on its own is not sufficient and is often overemphasized at the expense of informal experiential learning. To understand why, let's consider five stages of learning:[3]

1. **Knowledge**—Basic knowledge of the concepts and the facts
2. **Comprehension**—Demonstrated understanding of concepts
3. **Application**—Ability to apply concepts in simple contexts
4. **Evaluation**—Ability to apply judgment on when and how to apply concepts to more complex contexts
5. **Innovation**—Ability to extend concepts to new contexts

All that can be expected from a training class is to convey basic knowledge and *maybe* to provide some practice in applying the concepts. Time and experience are needed for the change to actually take hold in the day-to-day activities of the team. Only when this occurs is the change successful.

[3] These are loosely adapted from Benjamin S. Bloom's *Taxonomy of Educational Objectives* (Boston: Allyn and Bacon, 1984).

To achieve sustainable results, change must be driven by doing real work. There are several reasons for this:

- People don't like to change, but they will change if they can see that change leads to positive outcomes. "Positive outcomes" means different things to different people. For management, it might mean more reliable schedules and improved delivery capability. For the people on the project, it might mean improved career development or improved quality of life. Successful change initiatives usually require the majority of stakeholders to achieve some positive result. The sooner positive results are shown, the sooner support for the change builds.

- Even positive changes are initially disruptive; few situations are so bad that change is not initially destabilizing. In addition, most changes take some time to produce results. While the changes are underway, a reserve of "good will" is gradually consumed until results become visible. The longer the change takes to show results, the more "good will" is consumed. While "good will" exists, the organization can afford to be patient, but after it is exhausted, impatience takes over and the change effort typically falls apart. The problem is that organizations that need to change the most typically have the lowest reserves of good will. As a result, it is essential that results be demonstrated throughout the change effort to maintain the momentum of change.

- Real change only occurs when people's daily habits change; habits change only with time and practice. As a result, change based on "telling and teaching" almost never succeeds. Real change requires "doing."

Achieving improved results is not inconsistent with introducing change. In fact, achieving improved results while doing real work is essential to achieving sustainable results and is at the heart of the iterative approach. At every iteration assessment, lessons will be learned and acted upon to

improve the project's performance and results. By its very nature, iterative and incremental development encourages learning by doing for everybody involved in the project, which enables the approach to support its own adoption and improvement.

The Role of Coaching

When a new way of working is introduced, it is important to support the transfer of knowledge and application of new ideas. Workshops are an effective way to do this because they enable people to learn in the context of doing real work, and they accelerate the rate at which people can be productive using the new techniques.

To feel confident in the change, people need to be able to focus on "doing" and not be distracted by "figuring out what to do." Experienced practitioners who lead people through the new behaviors, getting them to apply the new process by doing, must provide guidance and coaching. Having experienced coaches available to the project team provides confidence in the outcome by providing experienced resources for the project team—people who have already been successful on other projects. A sense of confidence in the outcome is essential to building resilience within the team so that team members can recover from the inevitable setbacks that occur in any change effort.

To facilitate experiential learning, coaches enable the team to focus on "execution" instead of process definition. In addition, there are several other benefits of this approach:

- It builds support for the change by creating real success early.
- It lets team members focus on "learning by doing" rather than sending them to classes and hoping they can put the knowledge into action on their own.
- It builds expertise within the team so that over time the team can support the new way of working without external help.
- It reduces the risk of failures, delays, and quality problems resulting from the team's learning by doing.

- It accelerates the learning process by eliminating much of the uncertainty, discussion, and trial-and-error associated with learning by doing.

Some coaching is usually necessary to enable project team success. This will vary from team to team, but typically follow-on workshops are used to develop skills needed to execute the iteration plan. For example, in the Inception phase, project teams will need to understand how to understand the problem and create a vision for the solution; a workshop led by an experienced facilitator is often the most effective way to make progress toward this goal.

Sometimes the best coaches are on your own team in the form of the more experienced team members. Encouraging team members to work together to build team skills that improve team results also reinforces team cohesiveness. This does not happen accidentally, however; you will need to encourage it and plan for it.

Using the Iteration Plan to Provide a Roadmap for Change

The most effective method for introducing change is to tie the improvement effort directly to the work being performed. The most effective way to do this is to use the project and iteration planning effort (which must be done anyway) to drive the introduction of the new techniques just ahead of their need.

We have found that doing this through a series of focused workshops and subsequent hands-on mentoring jumpstarts results and facilitates skills transfer. For planning evolutions, an initial full-day workshop to bootstrap the development plan and the initial iteration followed by half-day workshops at the start of each iteration to review results from the previous iteration and plan activities in the upcoming iteration has proven to be most effective. The structure and contents of these workshops is as follows:

- Early in the evolution, a workshop is needed to create the development plan and the initial iteration plan and to outline

the approaches to be adopted to requirements and change management (often in the form of requirements management and change management plans). Iteration plans are defined so that just enough process is introduced to meet the objectives of the iteration and support the goals of the current "improvement cycle." In this way, the project team learns the new ways of working by applying them.

- At the transition between iterations, a workshop is needed to review iteration results, plan the next iteration, and identify issues and action plans. This can include making mid-project adjustments to the process used by the project, adjusting approaches as well as the pace of change. Sometimes the change will be too slow and the pacing can be accelerated; other times the project team will feel that the change was too fast and the improvement plans for future iterations will need to be scaled back.

The key to making improvements is to use the iteration plan itself to map the change activities *as well as* regular project activities. Change activities need to be viewed as integral to the project—tasks that the project does to achieve better results, not something external that detracts from results. If we don't believe that improved results will be derived from a change, why would we want to pursue it?

The iteration plan can be a powerful reinforcement to the change—or the means by which the change is undone: if the plan supports the change, it will tend to succeed, but if it fails to reinforce the new desired behaviors, the change will not happen no matter how much support is otherwise given.

As we have discussed, the project is structured into a series of iterations that establish a kind of heartbeat for the project, enabling it to set intermediate milestones to check progress, to establish points at which non-essential requirements can be scoped out, and to enable mid-course corrections to the project plan. This also acts as the heartbeat for change, enabling 1) short-term improvement objectives to be set, implemented, and assessed, 2) the change to be tested, and 3) mid-course corrections to made to the improvement plans.

Finally, planning is nothing if the execution of the plan is poor. Some people act as if a good plan will implement itself. In reality, a mediocre plan with excellent execution is better than an excellent plan with mediocre execution every time.

Conclusion

Adopting an iterative and incremental development approach is a fundamental change in working practices for the management team and everyone else involved in the project. Successful iterative and incremental development requires a progressive and adaptive approach to be taken to the management of the project and requires the whole team to embrace change and the continual improvement that this change will hopefully produce.

In any change effort, it is essential to demonstrate the value of the change as soon as possible to overcome resistance and build support for the change. The only way that can be done is by achieving the desired technical and business results quickly and efficiently. The fastest way to reach these results is to introduce the change as part of getting real work done; if the change is considered separate from the "real work," it will never produce results. With the guidance and leadership of an effective coach, and with the support of management to measure and reward positive results and positive change, teams can improve their process while getting real work done. Process improvement and getting results should not be considered mutually exclusive.

To expand beyond individual projects, you will need enlightened but benevolent dictatorship coupled with the demonstration through real results to all involved that the future can be better. It also requires leadership, real leadership—not the phony slogans of motivational posters, but roll-up-your-sleeves, hands-on leadership from the front that shows that you have a stake in the outcome. No one is going to believe you if you sit on the sidelines cheering; you have to be in the game.

Iterative development is not hard, but changing the way that people work is. In this book we have provided you with the background information and the practical guidance necessary to deliver better results through your software development efforts. The next step is yours: you now get to put these concepts into action. We hope that the approaches and techniques we have presented in this book will help you and your organization to succeed and thrive by achieving the full promise of iterative development.