What is iterative development?
(Part 2: The Customer Perspective)

Ian Spence, Chief Scientist, Ivar Jacobson Consulting, UK
Kurt Bittner, Product Strategy, IBM Rational software
15 Apr 2005

What does it mean for a project to work in an iterative and incremental manner? In this series of articles, we address this question by examining the various perspectives of those involved with the project. In the last article, we defined what we mean by iterative and incremental development and examined what this way of working means to the core development team producing the software. In this article, we focus on what it means for customers when a project chooses to work in an iterative and incremental manner. The series will draw to a close next month when we consider the effect of iterative and incremental development on the projects’ management teams.

To achieve the full promise of iterative and incremental development, you must ensure that its adoption will affect more than just the technical and development communities. In fact, adopting iterative practices ideally has a profound and lasting impact on all business people involved in the project. It also fundamentally changes the way these people specify, pay for, and realize the business benefits to be derived from the successful development of software solutions.

If the adoption of iterative and incremental development has no impact upon the business, and the way that it realizes the benefits provided by the iteratively developed solution, then its adoption becomes just a technical assurance issue and is of little consequence outside the development team itself. In order to deliver on its full potential, the approach must change the way the project and its stakeholders interact as well.

Most of the literature on iterative and incremental development focuses on the developer and development team lead. While the impact upon the development team is important, the true value of iterative and incremental development lies in the potential to dramatically improve business results. To achieve these benefits, the business -- including 1) the customer representative, 2) the business analyst team, 3) the end users of the system, and 4) the business leaders (sponsors) -- must become an active participant in the project and will need to change the way that it interacts with the development team. We will consider these four constituents of the customer perspective in this article, and how the changes represented by iterative and incremental adoption positively impact the business as a whole.

The customer representative

The adoption of an iterative and incremental approach by a software development project team profoundly affects the way the customer representative interacts with the project development team. As shown in Figure 2-1, each iteration will present developers with the opportunity to elicit objective feedback on the appropriateness of the solution and the business impact that it is likely to have. By basing the feedback on demonstrations of working versions of the system, the communication is kept focused and to the point, allowing the development to converge upon a solution that meets the system's essential minimal characteristics.
What is Iterative Development? By Spence & Rittner

Each iteration also gives the customer representatives an opportunity to comment on the work in progress and influence the future direction of the development. Once they review a working release that illustrates the implications of their requirements decisions, and the development team's interpretation of these requirements, the customer may decide to change the direction of the project or the prioritization of the remaining requirements. Again, by demonstrating small amounts of working functionality, the team can focus on what would constitute an accurate business solution and what the minimum amount of functionality to achieve this would be. At each iteration boundary, customer representatives have the opportunity to accept or reject the release produced by the development team and to adjust the direction in which the project is progressing. This empowers the customer team to become more involved in the project and to ensure that the solution will deliver the expected business benefit.

The rapid cycle of development, demonstration, and assessment provides many opportunities for the customer representatives to become involved in the project. This iterative approach allows them to:

- Take a more active and objective role in directing the project, actively steering the project as it progresses to ensure that everyone involved in the project converges on a shared understanding of what would constitute an appropriate business solution.
- Adopt a relationship with the project that is suitable for the amount of risk and trust there is in the relationship between collaborating parties.
- Act upon any lessons to be learned from their experiences to date -- an opportunity afforded to the development team as a whole.
- Maintain an objective assessment of the project's risks and progress to date.
- Build confidence in the team's ability to deliver

Regular customer demonstrations also benefit the development team, giving them increasing levels of:

- Understanding of the project as a whole
- Evidence of their convergence on an acceptable business solution
- Confidence in the acceptability of their plans
- Realistic expectations regarding the customer's requirements

The result is that the development project will no longer be a black hole into which money is poured in the hope that some software will appear at some vaguely specified point in the future. If the promised

convergence and understanding does not result from the initial set of iterations, then action can be taken to either re-orient the project or, if necessary, cancel it. This provides the business with an objective way of monitoring and adjusting its investments.

The business analyst

There is some disagreement in the industry about whether the requirements work should be part of the iterative process or not. In our experience, the true benefits of iterative and incremental development are only achieved when all team members, including those representing the interests of the business, are active participants in the project. This means that the requirements process needs to be tightly integrated with, and executed to the same timetable of, the iterative development of the solution.

Consider the following observations:

1. There is no value in documenting requirements that won't be delivered. It is difficult enough to deliver a solution on time and within budget when you focus only on the work that absolutely needs to be done to achieve this. Spending time developing things that will not be delivered is a distraction and puts the project at risk. Rather than assuming this additional work "will help the next project," we suggest that the work be left to the project that it will benefit. Bear in mind that when a first project does not meet its objectives -- a possibility heightened by unnecessary labor -- there is often no "next project."

2. Developers do not need to have all the requirements available before they start work. They just need enough requirements to drive their current work (in this case, the current iteration).

There is an awful lot of work that can be done on a system before all the requirements have been completed, since not all requirements are inter-related or of equal importance. A development team can easily go off and solve the problem of how the system will support 1,000 concurrent users long before all the user interfaces are designed. In fact, the design of the user interfaces typically has little or no effect on the scalability of the solution.

3. Requirements work tends to expand to fill the space available. On most projects, the ability for people to dream up new requirements easily outstrips the ability for any team to deliver them within the time and budget allotted.

Parkinson's law\(^1\), "Work expands so as to fill the time available for its completion," has never been truer than for requirements-related work. We have never seen a requirements team declare that they have found the full set of requirements before their allocated time has run out. In cases where the requirements team actually finds a sufficient set before the time runs out, they invariably go on to invent more requirements rather than declare the task complete.

Defining requirements takes almost no effort compared to the effort needed to implement them. The real art in defining requirements is not to identify as many as possible, but to find the minimum set of requirements needed to satisfy the business problem. More requirements are not better, and many a solution has been ruined by having too many features.

One of the basic assertions behind all iterative and incremental software development processes is that the only thing that matters is the delivery of a solution; anything that does not contribute to the delivery of a solution is superfluous and needs to be eradicated. The purpose of an iteration is to produce a working version -- albeit a partial version, often -- of the solution. To this end, the requirements to be implemented by the iteration will have to be known. This does not mean that all the requirements that will ultimately be satisfied need to be known before the development iterations start. Instead, we strive to have just enough understanding of the requirements to set a meaningful objective for the current iteration.

What is Iterative Development? By Spence & Rittner

There are several models for development of the requirements that drive an iterative development project. Let's consider four common models: waterfall, forward-loaded requirements/backward-loaded development, requirements pipeline, and just-in-time requirements.

**Waterfall requirements, iterative solution development**

In this case, all the requirements are defined up front before the iterative development of the solution starts. This pattern is shown in Figure 2-2.

![Figure 2-2: Development iterations with requirements defined up front](image)

As you probably realize by now, this approach is less than ideal because it usually results in the definition of some requirements that are never implemented, or the over-specification of some of the requirements that are.

This leads initially to the generation of unrealistic expectations and then an atmosphere of antagonism and disappointment as the realization that every requirement cannot be implemented starts to dawn on the parties involved.

It also leads to the requirements team working in isolation, depriving them of feedback from the development team, which in turn tends to produce conflict between the requirements team and the development team. It also prevents the lessons derived from the implementation of the first set of requirements from being learned by the team as a whole and used to improve the quality of the requirements and the whole team's effectiveness. This pattern is often seen in organizations where:

- The development team wishes to iterate but has yet to prove the appropriateness of the approach to the business community. This is how many organizations embark on the road to a more collaborative, iterative way of working.
- The development is being outsourced by a company unaware of iterative and incremental techniques but the outsource company itself has adopted them.
- The business has an adversarial relationship with the development team and does not trust the team to make any decisions or to work collaboratively with them.
- The business cannot prioritize the requirements and insists on treating them as a complete, indivisible set.

This requirements method must be approached with caution as it typically leads to an organizational structure that places team members into functional "silos" based on the type of work that they do. This creates barriers between team members, reducing communication and the teams' effectiveness in delivering solutions. This problem is usually compounded by the measurement systems that are put in place to reward people for doing work that is necessary but not sufficient on its own to deliver a successful solution. It is common for functionally organized teams to consider "success" to be the delivery of a good requirements document, or a signed-off software architecture document. While these things may be necessary for reaching the goal of delivering a solution, they are not valuable in and of themselves. If the solution fails to be delivered, the completed requirements document has yielded no value. While it may be necessary to organize reporting relationships along

functional lines to promote skills development and career growth, the measurement of progress and results needs to be aligned with the delivery of the solution. This usually leads to a “matrixed” reporting structure in which everyone reports to at least two organizations: one functionally aligned for career growth and the other aligned with the project and its goals.

**Forward-loaded requirements, backward-loaded development**

If there is any stability in the requirements then, as shown in Figure 2-3, the development iterations can start well before the requirements work is completed.

![Figure 2-3: Overlapping requirements specification and development iterations](image)

This appears to be a more iterative and incremental approach, but it still leaves the requirements and development teams working in different managerial silos with all of the communication overhead and conflict that this can entail.

**The requirements pipeline**

There is a temptation to move to a pipeline model where, as shown in Figure 2-4, the requirements team works on the requirements for the next iteration while the development team implements the requirements defined for this iteration.

![Figure 2-4: The requirements pipeline](image)

The questions this pattern immediately raises include: Which iteration is the team working on? Are the iterations overlapping? Is the team working on more than one iteration at a time? From a personal and project management perspective, how will people know which iteration they are working on or
what their priorities are? (Considering iteration from a management perspective will be the topic of the third installment in this series.)

The requirements pipeline pattern is really only suitable where it is impossible (for legal, geographical, or organizational reasons) for the requirements team and the development team to work together as a single team.

**Just-in-time requirements**

Ideally, to provide the most agile, flexible, and focused development team, the requirements and development work is fully integrated into a single set of well-defined iterations. This would extend the typical developers' model of iterative development to include the requirements specification of the release and its system testing. This would extend the model of iteration shown in last month's article to that shown in Figure 2-5.

This way of working is the most effective, but does depend on having an integrated, collaborative team that includes representation from the business and customer perspective, as well as that of the technically oriented development team.

---

**Figure 2-5: Iteration with integrated requirement specification and system testing**

This approach takes the principles of iterative, and incremental development to their natural conclusion by creating a single team focused on the effective, iterative, and incremental development of the entire business solution. This allows the team to adjust, and self-correct, its course and efforts to ensure the delivery of business value as quickly as possible.

As you can see, the impact on the business analyst team of adopting an iterative and incremental approach can be quite dramatic, as team members will find themselves an integral part of the project team, working on the project for its full duration and not just in its initial phase. Analysts will also find themselves defining requirements iteratively, often in a just-in-time fashion within the same iteration as they are implemented.

The analyst team will share the responsibility for delivering the solutions that they specify, and they will participate at a depth of involvement in the project required to fulfill that responsibility. The
requirements documentation, rather than being an immutable record of the "final" requirements that must be delivered, becomes a living contract between the business and the development team regarding what needs to be delivered to achieve a satisfactory resolution. When thought of as a contract, the requirements documentation infers obligations on both sides and is open for negotiation. Its scale and complexity will indicate the amount of trust existing between the two sides.

The system user

The system user perspective is very similar to that of the customer representative, in that users will get to see periodic evidence of progress to ensure that the solution will meet their real needs.

It is generally advantageous, to the project as a whole, to involve users in the demonstration and assessment of the releases produced by the iterations. This is unlikely to be necessary for every iteration, because allowing broad review of some of the early, partial implementations can be counter-productive. Regular interaction with system users also helps remind the development team of its responsibility to deliver business value and not just sexy new technology. It also enables the requirements themselves to be verified as well as the system being verified against them.

The benefits of soliciting early user feedback, especially in the areas of usability and user-interface design, is illustrated by the story in the sidebar, at right. We think it offers a valuable lesson.

The business leaders (or sponsors)

From the perspective of business leaders, the value of iterative development lies in its capability to support the considered incremental commitment of resources to the project and to balance the investment in the project against the project risk and its chances of success. The funding exposure to the project can be limited to the current iteration. If the progress and risk reduction demonstrated by an iteration is insufficient, then the project can be cancelled, or redefined, without great loss to the organization. The funding of an iteration can be seen as the purchasing of additional information about the project before committing to further investment toward its completion.

Barry Boehm uses the analogy of stud poker to illustrate the sponsors perspective on an iterative and incremental development project. In a stud poker game, you can put a couple of chips in the pot and receive two cards, one hidden and one exposed. You can also see the other players' exposed cards. If your cards don't promise a winning outcome, you can drop out of the game without great loss. If your two cards are both aces, then you can bet heavily on a winning outcome. In any case, you can decide during each round whether to gamble more of your chips to buy more information about your prospects of a win, or to fold based on what you know already.

Similarly, with each iteration more resources are being gambled on the successful outcome of the project and more information is available about the project's chances of success. If the project is successfully iterating and producing objectively measurable, proven increments, then the sponsors' confidence in the project will also increase incrementally and investment will continue. If, during the initial iterations, the project looks as though it has been dealt a losing hand, and faces insurmountable levels of risk, then the sponsors can drop out without great loss.

The concept of using the iterations in this way to manage project risk is fundamental to the successful adoption of iterative and incremental techniques.

Conclusion

Iterative and incremental development offers businesses many benefits above and beyond those proffered by the more traditional, waterfall development approaches. These include increased predictability, reduced time to market, higher-quality solutions, increased project agility, and increased productivity. The majority of these benefits will only be realized if the business itself starts to take a
What is Iterative Development? By Spence & Rittner

more active role in the projects and their iterations. This will fundamentally change the way businesses work with projects, affecting everything from the way that the requirements are captured to the way progress is assessed and, possibly most importantly, the way that the projects themselves are funded.

Next month, we will conclude this series by examining the effect of adopting iterative and incremental development practices on the project management team.

Notes
1 C. Northcote Parkinson, Parkinson's Law (1958)

Sidebar
Solicit user feedback -- early!

There was once an insurance company that appointed a visionary new IT director. The new IT director's first actions were to decree that:

• All new applications developed by the company would be designed and developed as Windows applications
• A PC-based Windows desktop was to be rolled out to all of the company's IT users
• All software would be developed using object-oriented techniques

The first major application to be developed under the new regime was a replacement for the old-fashioned, "green screen" system that was used in the company's call centre. The development went well, and the application was available for system testing on time and on budget. The visionary IT director was very impressed by the application, especially when its new user interface was demonstrated. The use of Windows -- with a "point-and-click" user interface, a sophisticated help system, and 3-D iconography -- was so elegant that the IT director was heard to remark, "Fantastic! With this point-and-click interface and extensive bubble help, even I could answer customer phone calls." Champagne corks popped as the team celebrated the success of the project and the new IT strategy.

In fact, the visionary IT director was not one of the people whose job it was to answer the phone calls. That was the job for a different set of people, who worked in very cramped conditions -- with no room for a mouse or a mouse pad on their desks. The last thing they wanted was a graphical user interface that took many mouse clicks to achieve what used to be achievable with a single hot key. Add to this the fact that no one was allowed to start work answering calls in the call centre without extensive training, rendering the need for the extensive bubble help redundant. Thus, the new system created a complete mismatch between the users' expectations and the physical and cultural conditions of the workplace.

The users were expecting a new system to make their lives easier, but they received one that only made their lives harder. So much effort had gone into engineering the new user interface that no other new functionality had been delivered. The system never made it through user acceptance testing and its initial pilot.

The moral of this sad tale: All this could have been easily avoided if early versions had been shared with members of the user community and if the development team had focused on delivering value to the business, rather than being seduced by the possibilities of playing with fashionable new technologies.