Roadtrip! A vacationer's guide to iterative development

Laura Rose, Quality Engineering Manager, IBM, Software Group
Initially published at http://www.ibm.com/developerworks/rational/library/jul06/rose/

Summary: from The Rational Edge: If you or someone you know is skeptical about the viability and value of iterative and incremental software development methods, perhaps what's needed is a familiar framework for understanding the basic concepts. Here, Laura Rose steps outside the realm of software to describe one form of iterative project: a cross-country roadtrip.

The iterative and incremental development process is built on the work of Barry Boehm's spiral model,[1] and it has been enhanced by many individuals and organizations, including the IBM Rational team, over the past two decades. The process is characterized by continuous discovery, invention, and implementation. With each iteration, the development team drives the project's artifacts to closure in a predictable and repeatable way.[2] The iterative approach accounts for changing requirements, it mitigates risks earlier, and it allows for tactical changes as developers learn along the way.

Even though iterative and incremental development methods have proved highly successful for both large and small development organizations, many still believe that iterative development does not work. While it is human nature to be skeptical of the unknown and untried, many, if not most, of us have implemented something very similar to the iterative development lifecycle in our life outside the software development arena. That is, we've taken a lengthy car trip across country -- many of us more than once.

This paper illustrates how iterative development techniques are used in everyday activities such as taking a family vacation.

Few would disagree that life involves continuous discovery, invention, and implementation. No one has their careers, goals, and accomplishments all plotted and well-defined at birth. We readily acknowledge that we learn as we grow, changing our strategies and even our goals with every experience and observation. Yet, when it comes to software development, we have difficulty applying those concepts. We tend to believe that a carefully laid-out plan and tight schedule is the only way to develop a new piece of software. In fact, iterative and incremental techniques offer a better approach to creating software; small steps in the process are accomplished, demonstrated, and evaluated, and discoveries often suggest new ways to go about achieving the next steps.

To illustrate this iterative process, let's compare it to something many of us are familiar with: managing a family vacation. Even if you've never taken a family vacation (or it's been a long time), I think you'll see the value of this cross-country trip metaphor.

The project

Let's pretend that our sister Jane is getting married on June 3 in Orlando, Florida, USA, and we live in San Diego. We have also agreed to pick up other family members along the way.

Already, as in an iterative development project, we have our mandatory requirements (get to Orlando!), release-defining features (pick up other relatives), and a fixed release date (our sister is getting married on June 3, whether we're there or not). Note that, in real life, we have no trouble accepting hard deadlines -- like wedding dates. How often have you heard someone complain that a selected wedding date is too aggressive? We simply accept the invitation, or we send our regrets.

Just as at the start of a software development project, we have multiple methods to accomplish our goals. We can fly, take a train, rent a car, or combine any of these options. We can take three days or three weeks to get there. And, as in a software project, we have other people (stakeholders) involved. Imagine that our stakeholders are Aunt Marian in Santa Fe, New Mexico; Uncle Cid in Baton Rouge, Louisiana; and cousin Tanya in Jackson, Mississippi.

Project features

At the start, everyone agrees that it would be fun to make this a three-week family vacation, with our primary stops at the three cities where our relatives live. To increase value and enjoyment, we plan some additional excursions en route. These additional excursions will be optional, depending on our progress, but they surely will be "nice to do."

So far, this trip partly resembles the Inception phase of an iterative development project: It's common to have a few release-defining (primary) features and several secondary or nice-to-do features. The release-defining feature set includes the mandatory and highly publicized components. As this title suggests, these features determine the critical path. By definition, our release date will be postponed if those selected features are not of the agreed quality or completeness. The development team continues to work on the secondary features within the same schedule. But we do not intend for the nice-to-do features to affect the release date. Because the secondary features are not publicized, we have the flexibility to reposition our resources away from those secondary features in order to assure that the primary feature set is completed on time.

The side-excursions (secondary vacation stops) are initially selected based upon the routes to the various family member pickup points. If time gets tight, these nice-to-see landmarks can be omitted to get us back on schedule. So, even though we don't know all the stops along the way, we do have a skeleton of the route (project plan), when we need to be at our various pickup points (our requirements, milestone dates, and iteration phases) and the wedding date (project end date).

We secure a friend's RV as our travel method. We think that this would be relaxing and comfortable. The RV adds flexibility, because we are not required to stay at hotels every night. But the RV does have limited space. So for this to be successful, we need everyone to conform to certain traveling criteria, such as luggage limits, a commitment to the team's expenses, what they need to supply for the trip, and our various timetables. If at any point along the journey, we don't meet our agreed-upon travel criteria, we need to stop, evaluate our situation, and adjust.

Legs of the journey

In the iterative development method, there are initial entry and exit criteria that govern whether each leg of our development journey can begin. These targets are used to guide and validate the stability, reliability, and accuracy of the product under development. These agreed-upon criteria are created at the beginning of the project, with the understanding that they can be appropriately altered along the way.[3] An example of an iteration exit or entrance criteria would be:

1. All planned features delivered for this iteration
2. All stakeholders reviewed and approved planned test coverage for this iteration
3. All planned tests executed, with 95 percent passing
4. No known blocking or high priority issues

We continually monitor the project according to these criteria along the way. If at any point along the development lifecycle, we recognize that we're off course, we stop, evaluate our situation, and adjust. It's also important to realize that "adjust" can mean modifying the entrance/exit criteria.

Because we will be at different points of our vacation at each iteration milestone (see Figure 1), the entrance and exit criteria for each leg of the journey will be different for each participant. As an example, you and I will be traveling for three weeks. Therefore, our luggage allocation would be larger than someone who is picked up on the last few days of the vacation.

At the start, we identify a general plan and important criteria for the trip. After a few modifications to the plan, everyone agrees to the terms, conditions, and timetable.

We're on our way.

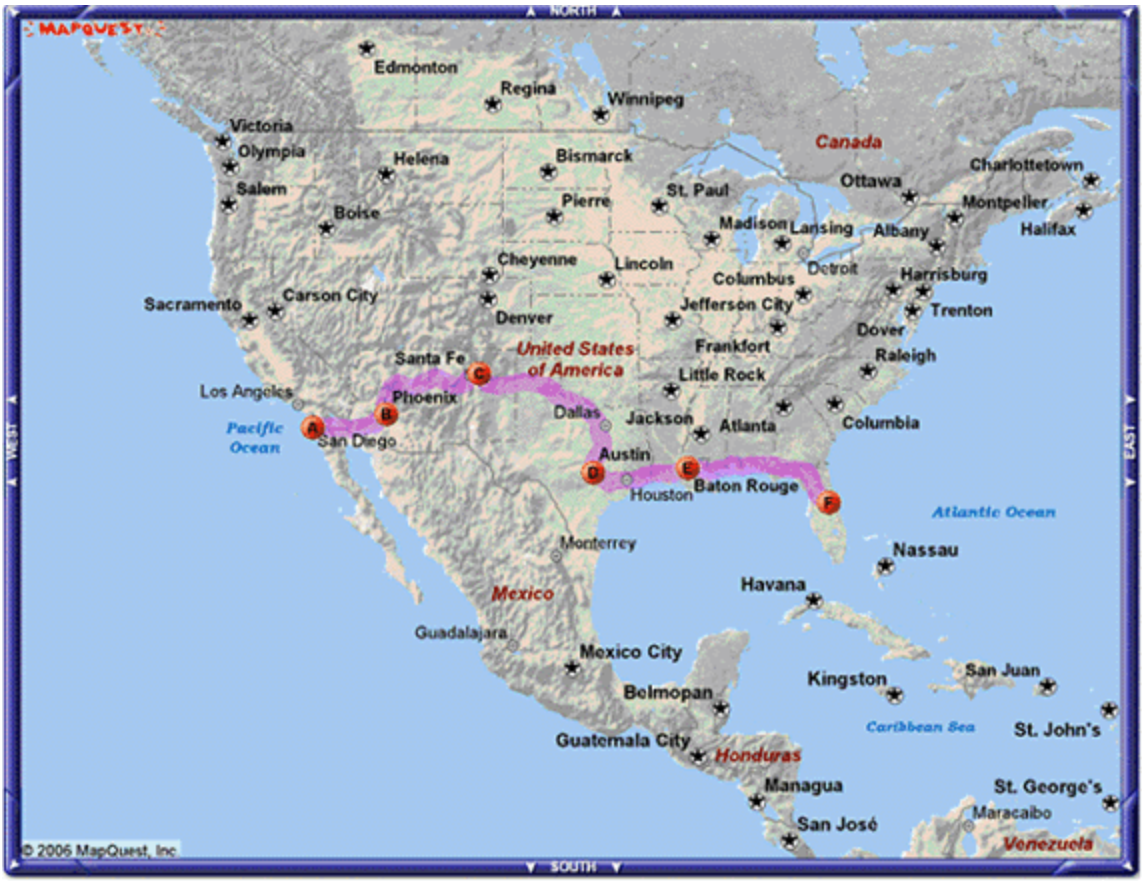

Figure 1: Three milestones before the end of our iterative trip: Santa Fe, Baton Rouge, and Jackson.

Although we know we need to make three stops on this trip, we're not too concerned with our third stop in Jackson just yet. As we begin the journey, we focus on getting to Aunt Marian in Santa Fe. We refine a more detailed route for that leg of the journey (the first iteration). Since we have friends in Phoenix, we pop in there for a brief visit. We aren't exactly sure what they have in store for us, but we know that they plan to take us sightseeing in their hometown. We had already called ahead with our timetable and constraints. Our friends in Phoenix schedule their activities to coincide with our requirements. We also make sure to periodically touch base with them on route.

Note that leaving the planning of the time in Phoenix to our friends is similar to contracting or outsourcing a feature or component in a development project. We establish certain timetables, constraints, checkpoints, and communication procedures; but we leave the details to the third party.

We have a great time in Phoenix and we successfully get to Santa Fe on time.

The first hiccup

When we arrive in Santa Fe, Marian is ready and we review our status. Marian meets the agreed-upon luggage criteria, she has budgeted the proper spending cash, and she's packed the items that were on her list of things to bring to the wedding. The only hiccup is that she is insisting on bringing her puppy, Dogma. We didn't originally plan for this, so we need to evaluate what type of adjustment is needed. Although you or I don't have a specific problem with that, we need to check with all the stakeholders (similar to conducting a change management review). We regrettably find out that Cousin Sid (the object of the next leg of our journey) is allergic to dogs. Dogma can't travel with us.

In many styles of software development, adding a feature after the project plan has been approved is termed as "feature creep." But in the iterative development lifecycle, we expect changes in requirements and requests. We intend to constantly learn from the previous iteration. Since progressive requirements refinement is one of the essential concepts in iterative development, conducting requirement reviews at the start or end of each iteration is critical.

In our case, Marian is disappointed, but she understands. Because she previously did a risk-assessment and realized that Dogma may not be allowed to come with us, her contingency plan was already in place. Her friend (several towns away) had already agreed to take the dog. Dropping off the dog will take us several hours off the planned route, so we review our day's schedule with this adjustment in mind. We decide that we can easily include the route change without any negative consequences. Marian's dog-sitting friend also offered to give us an expert sightseeing tour of that area. The consequence of the change was trivial and even beneficial: i.e., because we adjusted to this change, we had a more exciting interactive tour (than the landmark previously planned).

In iterative development, this is much like substituting one optional feature for another as we continually learn about our project and our customer needs. Even though the gathered knowledge from the previous iteration might change the contents or course of the next iteration, our primary goals and end date stays stable.

We successfully deliver Dogma. We have a great time meeting and visiting with Darla. We also learned that Darla's son and daughter will be attending the same high school as our son Josh. We made some future plans to travel together for different school events and even investigate the possibility that the children might room together. To sum up, we are able to make some tentative plans about the future, all because the flexibility built into the iterative process allows us some deviation from the original plan.

An actual setback

Our next major iteration milestone is Cousin Sid in Baton Rouge. On the way, we want to stop at three cities in Texas: Austin, San Antonio, and Houston. We spend several enjoyable days at Austin and San Antonio. Everything is going to plan until the RV breaks down outside of San Antonio. It's unknown exactly what it will cost or how long it will take to fix the RV. The optimistic estimate is a few days, but the mechanic can't tell us for sure until he spends more time with the RV (much like investigating a defect in our code). Unfortunately, we didn't foresee something like this occurring. It has the potential for setting us back several days.

In any software development project, the team tries to identify risks, the probability of occurrence, impact to the overall project, and contingency plans. In iterative development cycles, risk management and identification are continuously reviewed in each iteration.

So, just as in a software project, we have a conference call with all the stakeholders. Since we don't know exactly when the RV will be fixed, we need a new plan. We can't change the end date: Jane is getting married whether we're there or not. If we stick to the current project plan, there's a high probability we will miss the wedding. Since Marian is the maid-of-honor, she is not pleased. She feels very strongly about taking immediate action to assure we get there on time.

We discuss several options. The plan we agree on is to have Tanya (in Jackson) fly to Baton Rouge to meet Sid. When the RV is fixed, we will go directly to Baton Rouge and pick up both of them at that iteration milestone. Eliminating the side excursion to Jackson gives us a seven-day buffer.

Tanya agrees to the new expense of her airfare. Of course, she isn't the only one bearing unforeseen expenses: We have the RV to repair, and everyone has a different opinion about how the new expenses should be handled. After some discussions, we eventually arrive at a solution that is acceptable to the team (this is similar to acquiring additional equipment and resources needs in a software project to recover from an unexpected setback).

Refining the contingency plan

Because the RV repair time is still unknown, Marian is nervous about making our final destination on time. So she suggests a refinement to better offset the risk: Limit the time we wait for the RV repair. For instance, if the repairs take longer than five days, she suggests we abandon the RV and use a different method of travel. In software development, this is called timeboxing the "wait" or "delay" cycle. This method allows us to control the unknown items, which eliminates the day-to-day float that takes most projects off schedule. Marian looks into various travel methods like plane, train, and bus. Considering everyone's budgets, travel schedules, comfort, and other concerns, she recommends that if the RV isn't drivable within five days, we take a train to Orlando. Tanya and Sid would leave from Baton Rouge, and we would go directly from San Antonio. I volunteer to stay with the RV, understanding that I might miss the wedding. So now we had an iterative and refined contingency plan for the risk that the RV may not be fixed in time.

Marian wonders if we should call and tell Jane about our current situation. Jane is much like our customer in a software development project, who is interested in the final delivery of our software application. After discussing it, we decided that Jane has too many other things to worry about. If we have to put our contingency plan into effect, we may want to give her a heads-up. But right now, we think we can contain these events. Any one of us can take the train (or even fly) directly to Orlando, at any time, decoupling the dependency to the rest of the family. Even though traveling together was an important "requirement" at the start of the project, we understand that sometimes priorities change.

In times of crisis, the requirements can be altered to fit the current situation or need. In this example, the requirement for all of us to get to the wedding remains intact. But because of current circumstances, we've changed it a bit -- i.e., it's no longer mandatory that we get there all together or at the same time. The final goal is still met, but it's accomplished in a slightly different manner. In an iterative development lifecycle, modification to requirements based on the current situation or need is not only acceptable, it's actually expected.

Marian agreed that if she felt anxious about the time, she could always go directly to Orlando on her own. In iterative development, this is similar to deciding how much change you can completely contain within your group before raising a red flag that affects external teams.

Now that we have a plan in place for our current situation and a contingency plan for our risk, we can relax a little. We won't have the RV for a few days, so we elect to find a nice bed and breakfast. The hosts are gracious, fun-loving, and knowledgeable about the area. They are sympathetic to our story and promise to make our few days of captivity as enjoyable as possible. We use this creatively, in ways that expand our knowledge.

Adjustments and opportunities

In iterative development, we often encounter delays in a project, which can give us time to reduce the defect inventory and stabilize the iteration's deliverables, and thus help keep the overall mission on track. Often delays occur while waiting for other parts of the design project to synchronize with the main branch. But these delays can present opportunities for other team members, who are given unexpected time to stabilize and reduce defects in the baseline code.

And wouldn't you know it? The RV is actually fixed in three days; we are on the road again, but because we're a little wary of the stability of the RV, we go directly to Baton Rouge. In fact, we arrive ahead of schedule, and Tanya's flight from Jackson doesn't arrive until the next day. So we visit with Sid until Tanya catches up with us.

At this point, we are pretty much at end-game. Let's take a closer look at the iteratively developed vacation events. As we look back over the progress of the trip, we can chart the iterations (both planned and actual) as shown in Figures 2 and 3. Figure 2 portrays the vacation plans at the start of our vacation project. Through constant strategy reviews, adaptability, and iterative task planning, our actual travel plans successfully resulted in Figure 3.

Iteration 1	
	Borrow Van
	Visit friends in Phoenix
	Pick up Marian in Sante Fe by May 5
Iteration 2	
	Other places
	Visit Austin
	Visit San Antonio
	Visit Houston
	Pick up Cid in Baton Rouge
Iteration 3	
	Visit Columbia
	Visit Hattleburg
	Pick up Tanya in Jackson
Iteration 4	
	Visit Tallahassee
	Visit Gainesville
	Visit Daytona Beach
	Arrive in Orlando by June 3
	Planned Happy Ending

Figure 2: Original iteration plan at start of vacation

Iteration 1	
	Borrow Van
	Visit friends in Phoenix
	Pick up Marian in Sante Fe by May 5
Iteration 2	
	Added side-trip to Darla's house to drop off dog.
	Removed other visits
	Visit Austin
	Visit San Antonio
	Have to fix RV
	Tanya Travels to Baton Rouge to meet Cid
	Wait for RV to be fixed in San Antonio
	Pick up Cid and Tanya together in Baton rouse
Iteration 3	
	Arrive in Orlando early
	Visited Disney World
	Actual Happy Ending

Figure 3: Actual iteration activities during vacation, adjusted from those shown in Figure 2

Continuous testing

Just like the previous pickup points (iteration milestones), we "test" to ensure that we've met the proper luggage limits and budget, and have all the items we need to bring to the wedding. And... ah ha! We find that, in the rush to secure alternative travel plans, Tanya forgot the items she was supposed to bring for the wedding. But thanks to our iterative testing at each iteration point, we caught this oversight before the final deadline (the wedding itself). Not a big problem. We do some manageable shopping to get Tanya back on track. (This illustrates the importance of continuous testing at each iteration and verifying our iteration entrance and exit criteria as our software project progresses.)

Now there is only the ten-hour trip to Orlando left. We take a leisurely drive, sightseeing at Mobile, Alabama, and Tallahassee, Florida. We eliminate the other side trips and arrive in Orlando ahead of schedule. Some of us spend a day at Disney World, while Marian uses her time for her bridesmaid duties.

The iterative adventure

As you can see in this simple example of iterative traveling, we've encountered similar events to software risks, feature creep, unexpected breakdowns beyond our control, and recovery plans. Just like our travel plans, iterative development and testing can be used effectively to increase the probability of meeting your software schedules.

Notes

[1] Barry W Boehm, "A Spiral Model of Software Development and Enhancement," IEEE Computer, May 1988, pp. 61-72.

[2] Philippe Kruchten, *The Rational Unified Process An Introduction*, pp. 7-8.

[3] Acceptance criteria, such as entrance and exit standards, should continually be reviewed and evaluated based on continuous knowledge and project refinement. It's even acceptable to alter them, if appropriate, based upon new learnings or course changes.

About the author

Laura Rose is a quality assurance expert and the product manager responsible for automated performance test tools at IBM Rational. In addition to leading projects in both software programming and testing environments, she has thirteen years of programming experience and ten in test management. She has been a member of the American Society for Quality, the Triangle Quality Council, and the Triangle Information Systems Quality Association. She is published and has presented at various test and quality conferences including IBM Test Symposium West, Practical Software Quality Conference (East and West), the American Quality Society Conference, Better Software Conference & EXPO and StarWest. You can reach her at llrose@us.ibm.com.