

From Waterfall to Iterative Lifecycle –

A tough transition for project managers

Philippe Kruchten

Rational Software White Paper
TP- 173 5/00

Table of Contents

Introduction	2
More planning work	2
<i>Trap: Overly detailed planning up to the end.....</i>	<i>2</i>
Acknowledging rework up-front	2
<i>Trap: Project not converging.....</i>	<i>3</i>
<i>Trap: Let's get started, we'll decide where to go later.....</i>	<i>3</i>
<i>Trap: Falling victim of your own success.....</i>	<i>3</i>
Putting the software first.....	3
<i>Trap: Focus on the wrong artifact.....</i>	<i>4</i>
Hitting hard problems earlier	4
<i>Trap: Putting your head in the sand.....</i>	<i>4</i>
<i>Trap: Forgetting about new risks.....</i>	<i>4</i>
Clashes of lifecycle models.....	4
<i>Trap: Different groups operating on their own schedule.....</i>	<i>5</i>
<i>Trap: Fixed-price bidding in inception.....</i>	<i>5</i>
Accounting for progress is different.....	5
Deciding on number, duration and content of iterations	5
<i>Trap: Pushing too much in the first iteration</i>	<i>6</i>
<i>Trap: Too many iterations</i>	<i>6</i>
<i>Trap: Overlapping iterations.....</i>	<i>6</i>
A good project manager and a good architect.....	7
<i>Trap: Use the same person as the PM and the architect</i>	<i>7</i>
Conclusion.....	7
Acknowledgements.....	8
Further Reading	8

Introduction

Although the Rational Unified Process™ (RUP) advocates an iterative or spiral approach to the software development lifecycle, do not believe for one second that the many benefits it provides come for free. Iterative development is not a magical wand that when waved solves all possible problems or difficulties in software development. Projects are not easier to set up, to plan, or to control just because they are iterative. The project manager will actually have a more challenging task, especially during his or her first iterative project, and most certainly during the early iterations of that project, when risks are high and early failure possible. In this paper, I describe some of the challenges of iterative development from the perspective of the project manager. I also describe some of the common “traps” or pitfalls that we, at Rational, have seen project managers fall into through our consulting experience, or from reports and war stories from our Rational colleagues.

More planning work

Iterative development does not necessarily mean less work and shorter schedules. Its main advantage is to bring more predictability to the outcome and the schedule. It will bring higher quality products, which will satisfy the real needs of end-users, because you have had time to evolve requirements and evolve a design and an implementation.

Iterative development actually involves much more *planning* and is therefore likely to put more burden on the project manager: an overall plan has to be developed, and detailed plans will in turn be developed for each iteration. It also involves continuous negotiation of tradeoffs between the problem, the solution and the plan. More architectural planning will also take place earlier. Artifacts (plans, documents, models and code) will have to be modified, reviewed and approved repeatedly at each revision. Tactical changes or scope changes will force some continuous replanning. Thus, team structure will have to be modified slightly at each iteration.

Trap: Overly detailed planning up to the end

It is typically wasteful to construct a detailed plan end-to-end, except as an exercise in evaluating the global envelope of schedule and resources. This plan will be obsolete before reaching the end of the first iteration. Before you have an architecture in place and a firm grip on the requirements — which occurs roughly at the LCA milestone — you cannot build a realistic plan.

So, incorporate precision in planning commensurate with your knowledge of the activity, the artifact or the iteration being planned. Near term plans are more detailed and fine grained. Far term plans are maintained in coarse-grained format.

Resist the pressure that unknowledgeable or ill-informed management may bring to bear in an attempt to elicit a “comprehensive overall plan.” Educate and explain the notion of iterative planning and the wasted effort of trying to predict details far into the future. An analogy that is useful: a car trip from New York to L.A. You plan the overall route, but only need detailed driving instructions to get you out of the city and onto the first leg of the trip. Planning the exact details of driving through Kansas, let alone the arrival in California, is unnecessary as you may find that the road through Kansas is under repair and you need to find an alternate route, etc.

Acknowledging rework up-front

In a waterfall approach, too much rework comes at the very end, as an annoying and often unplanned consequence of finding nasty bugs during final testing and integration. Even worse, you discover that most of the cause of the “breakage” comes from errors in the design, which you attempt to palliate in implementation by building workarounds that lead to more breakage.

In an iterative approach, you simply acknowledge upfront that there will be rework, and initially a lot of rework: as you discover problems in the early architectural prototypes, you need to fix them. Also, in order to build executable prototypes, stubs and scaffolding will have to be built, to be replaced later by more mature and robust implementations. In a healthy iterative project, the percentage of scrap or rework should diminish

rapidly; the changes should be less widespread as the architecture stabilizes and the hard issues are being resolved.

Trap: Project not converging

However, iterative development does not mean scrapping everything at each iteration. Scrap and rework has to diminish from iteration to iteration, especially after the architecture is baselined at the LCA milestone. Developers often want to take advantage of iterative development to do gold plating: introducing yet a better technique, to perform rework, etc. The project manager has to be vigilant as to not allow rework of elements that are not broken, that are OK, or are good enough. Also, as the development team grows in size and as some people are moved around, newcomers are brought in. They tend to have their own ideas on how things should have been done. Similarly, customers (or their representatives in the project: marketing, product management) may want to abuse the latitude offered by iterative development to accommodate changes, and/or to change or add requirements with no end. This effect is sometimes called *Requirements Creep*. Again, the project manager needs to be ruthless in making tradeoffs and in negotiating priorities. Around the LCA milestone, the requirements are baselined, and unless the schedule and budget are renegotiated, any change has a finite cost: getting something in means pulling something out.

And, remember that “perfect is the enemy of good.” (Or in French: “le mieux est l’ennemi du bien”).

Trap: Let’s get started, we’ll decide where to go later

Iterative development does not mean perpetually fuzzy development. You should not simply begin designing and coding just to keep the team busy or with the hope that clear goals will suddenly emerge. You still need to define clear goals, put them in writing and obtain concurrence from all parties; then refine them, expand them, and obtain concurrence yet again. The bright side is that in iterative development, you need not have all the requirements stated before you start designing, coding, integrating, testing and validating them.

Trap: Falling victim of your own success

An interesting risk comes near the end of a project, at the moment the “consumer bit” flips. By this we mean that the users go from believing that nothing will ever be delivered to believing that the team might actually pull it off. The good news is that the external perception of the project has shifted: whereas on Monday they would have been happy if *anything* were delivered; on Tuesday, they become concerned that not *everything* will be delivered. This is the bad news. Somewhere between the first and second beta, you find yourself inundated with features people are concerned with making into the first release. Suddenly, these become major issues. The project manager goes from worrying about delivering minimal acceptable functionality to a situation where every last requirement now becomes “essential” to the first delivery. It is almost as though when this bit flips, all outstanding items get elevated to an “A” priority status. The reality is that there is still the same number of things to do, and the same amount of time with which to do them. While external perception may have changed, prioritization is still very, very important.

If at this crucial moment the project manager loses his nerve and starts to cave in to all requests, he actually puts the project in schedule danger again! It is at this point that he or she must continue to be ruthless and not succumb to new requests. Even trading off something new for something taken out may increase risk at this point. Without vigilance, one can snatch defeat from the jaws.

Putting the software first

In a waterfall approach, there is a lot of emphasis on “the specs” (i.e., the problem-space description) and getting them right, complete, polished and signed-off. In the iterative process, the *software* we develop comes first. The software architecture (i.e., the solution-space description) needs to drive early lifecycle decisions. Customers do not buy specs; it is the software product that is the main focus of attention throughout, with both specs and software evolving in parallel. This focus on “software first” has some impact on the various teams: testers, for example, may be used to receiving complete, stable specs, with plenty of advanced notice to start testing; whereas in an iterative development, they have to begin working at once, with specs and requirements that are still evolving.

Trap: Focus on the wrong artifact

“I am a project manager, so I should focus on having the best set of management artifacts I can; they are key to everything.” Not quite true! Although good management is key, the project manager must ensure in the end that the final product is the best that can be produced. Project management is not an exercise in covering yourself by showing that you have failed *despite* the best possible management. Similarly, you may focus on developing the best possible spec because you have been hurt by poor requirements management in the past; this will be of no use whatsoever if the corresponding product is buggy, slow, unstable and brittle.

Hitting hard problems earlier

In a waterfall approach, many of the hard problems, the risky things, and the real unknowns are pushed to the right in the planning process, for resolution in the dreaded system integration activity. This leaves the first half of the project as a relatively comfortable ride, where issues are dealt with on paper, in writing, without involving many stakeholders (testers, etc.), hardware platforms, real users or the real environment. And then suddenly, the project enters integration Hell and everything breaks loose. In iterative development, planning is mostly based on risks and unknowns, so things are tough right from the onset. Some hard, critical and often low-level technical issues have to be dealt with immediately, rather than pushed out to some later time. In short, as someone once said to me: in an iterative development you cannot lie (to yourself or to the world) very long. A software project destined for failure should meet its destiny earlier in an iterative approach.

One analogy is of a university course in which the professor spends the first half of the semester on relatively basic concepts, giving the impression that it is an easy class that allows students to receive good marks at the mid-term with minimal effort. Then suddenly, an acceleration occurs as the semester comes to a close wherein the professor tackles all the challenging topics shortly before the final exam. At this point, the most common scenario is the majority of the class buckles under the pressure, performing lamentably on the final exam. It is amazing that otherwise intelligent professors are amazed at this repeated disaster, year after year, class after class. A smarter approach would be to front-load the course, tackling 60% of the work prior to the mid-term, including some challenging material. The correlation to managing an iterative project is to not waste precious time in the beginning solving non-problems and accomplishing trivial tasks. The most common reason for *technical* failure in startups: “They spent all their time doing the easy stuff.”

Trap: Putting your head in the sand

It is often tempting to say: “This is a delicate issue, a problem for which we need a lot of time to think. Let us postpone its resolution until later, which will give us more time to think about it.” The project then embarks on all the easy tasks, never dedicating much attention to hard problems. When it comes to the point where a solution is needed, hasty solutions and decisions are taken, or the project derails. You want to do just the opposite: tackle the hard stuff immediately. I sometimes say: “If a project must fail for some reason, let it fail as soon as possible, before we have expended all our time and money.”

Trap: Forgetting about new risks

You performed a risk analysis in inception and used it for planning, but then forgot about risks that develop later in the project. They come back to hurt you later. Risks should be re-evaluated constantly, on a weekly, if not, monthly basis. The original list of risk you developed was just tentative. It is only when the team starts doing concrete development (software first) that they will discover many other risks.

Clashes of lifecycle models

The manager of an iterative project will often see clashes with his environment and other organizations such as top management, customers, and contractors, who have not adopted or even understood the nature of iterative development. They expect completed and frozen artifacts at key milestones, they do not want to review requirements in small installments, they are shocked by rework, and they do not understand the purpose or value of some ugly architectural prototype. They perceive iteration as just fumbling purposelessly, playing around with technology, developing code before specs are firm, and testing throwaway code.

At minimum, make your intentions and plans clearly visible. If the iterative approach is only in your head and on a few whiteboards shared with your team, you will run into trouble later on.

The project manager must protect the team from external attacks and politics in order to prevent the outside world from disrupting or discouraging the team. He or she must act as a buffer. In order to be “the steady hand on the tiller,” the project manager must build trust and credibility with the external community. Therefore, visibility and “tracking to plan” is still important, especially in light of “the plan” being somewhat unconventional in many people’s eyes. In fact, it is actually *more* important.

Trap: Different groups operating on their own schedule

It is better and easier to have all groups (or teams, or subcontractors) operating against the same phase and iteration plan. Often project managers see some time-optimization in fine-tuning the schedule of each individual team, each of which ends up having its own iteration schedule. All the perceived benefits will be lost later and teams will be forced to synchronize to the slower group. As much as feasible, put everybody on the same heartbeat.

Trap: Fixed-price bidding in inception

Many projects are pushed into bidding for a contractual development far too early, somewhere in the middle of inception. In an iterative development, the best point in time for all parties to do such bidding is at the LCA milestone (end of elaboration). There is no magic recipe here: it takes some negotiation and education of the stakeholders, showing the benefits of an iterative development, and eventually a two-step bidding process.

Accounting for progress is different

The traditional earned-value system to account for progress is different, since artifacts are not complete and frozen, but are reworked in several increments. If an artifact has a certain value in the earned value system, and you get credit for it at the first iteration where you created it, then your progress is over-optimistic. If you only get credit when it becomes stable two or three iterations later, your measure of progress becomes depressingly pessimistic. So when using such an approach to monitor progress, artifacts must be decomposed in chunks. For example: initial document (40%), first revision (25%), second revision (20%), final document (15%), and each chunk must be allocated a value. You can then use the earned value system without having to complete each element.

An alternative would be to organize the earned value around the iterations themselves, and gauge the value from the evaluation criteria. Then the intermediate tracking points (usually monthly) reported in the Status Assessment would be built around the Iteration Plan. This requires a finer-grain tracking of artifacts than the traditional requirements spec, design spec, etc. because you are tracking the completion of various use cases, test cases and so on.

As Walker Royce says, “A project manager should be more focused on measuring and monitoring *changes*: changes in requirements, in the design, in the code, than in counting pages of text and lines of code. And Joe Marasco adds, “Look out not only for change, but also for churn. Things that change multiple times to return to the same starting point are a symptom of deeper problems.”

On the positive side, having concrete software that runs early can be used by the wise project manager to obtain some early credibility points, to show off progress in a more meaningful fashion than just completed and reviewed paperwork and hundreds of check-boxes ticked off. Also, engineers prefer “demonstrations of how it works” rather than “documentation of how it should work.” Demonstrate first, then document.

Deciding on number, duration and content of iterations

What do we do first? The manager who is new to iterative development often has a hard time deciding on the content of iterations. Initially, this planning is driven by risk, technical and programmatic, and by criticality of the functions or features of the system under construction. (The RUP gives guidelines for deciding the number and duration of iterations.) The criteria also evolves throughout the lifecycle. In construction, planning is geared to completing certain features or certain subsystems; in transition, to fixing problems and increasing robustness and performance.

Trap: Pushing too much in the first iteration

We talked above about not tackling the hard problems first. On the other hand, going too far in the opposite direction is also a recipe for failure. There is a tendency to want to address too many issues and cover too much ground in the first or first few iterations. This fails to acknowledge other factors: a team needs to be formed (trained), new techniques need to be learned, and new tools need to be acquired. And often, the problem domain is new to many of the developers. This often leads to a serious overrun of the first iteration, which may discredit the entire iterative approach. Or, the iteration is aborted — declared done when nothing runs — which is basically declaring “victory” at a point where none of the lessons may be drawn, missing most of the benefits of iterative development.

When in doubt, or when confronted with crisis, make it smaller somehow (this applies to the problem, the solution, the team). Remember that completeness is a late lifecycle concern. “Appropriate incompleteness” should be the manager's early lifecycle concern. If the first iteration contains too many goals, split it into two iterations, and then ruthlessly prioritize which objectives to attempt to achieve first.

It is better to shoot for a simpler, more conservative goal early in the project. Note we didn't say *easy*. Having a solid, acquired result early in the process will help build morale. Many projects that miss the first milestone never recover. Most that miss it *by a lot* are doomed despite subsequent heroic efforts. Plan to make sure you don't miss an early milestone by a lot.

Trap: Too many iterations

First, a project should not confuse the daily or weekly builds with iterations. Since there is fixed overhead in planning, monitoring and assessing an iteration, an organization that is unfamiliar with this approach should not attempt to iterate at a furious rate on its first project. The duration of an iteration should also take into consideration the size of the organization, its degree of geographic distribution, and the number of distinct organizations involved. Revisit our “six plus or minus three” rule of thumb.

Trap: Overlapping iterations

Another very common trap is to make iterations overlap too much. Starting to plan the next iteration somewhere towards the last fifth of the current iteration, while attempting to have a significant overlap of activities (i.e., starting detailed analysis, designing and coding the next iteration before finishing the current one and learning from it) may look attractive when starring at a GANTT chart, but will lead to problems. Some people will not be committed to following up and completing their own contribution to the current iteration; they may not be very responsive to fixing things; or they will just decide to take any and all feedback into consideration only in the next iteration. Some parts of the software will not be ready to support the work that has been pushed forward, etc. Although it is possible to divert some manpower to perform work unrelated to the current iteration, this should be kept minimal and exceptional. This problem is often triggered by the narrow range of skills of some of the organization's members, or a very rigid organization: Joe is an analyst, and this is the only thing he can or wants to do; he does not want to participate in design, implementation or test. Another negative example: A large command and control project has its iterations so overlapped that they are basically all running in parallel at some point in time, requiring management to split the entire staff across iterations, with no hope of feeding back lessons learned from the earlier ones to the later ones.

See figure 1 for a few common “anti-patterns” of iteration planning.

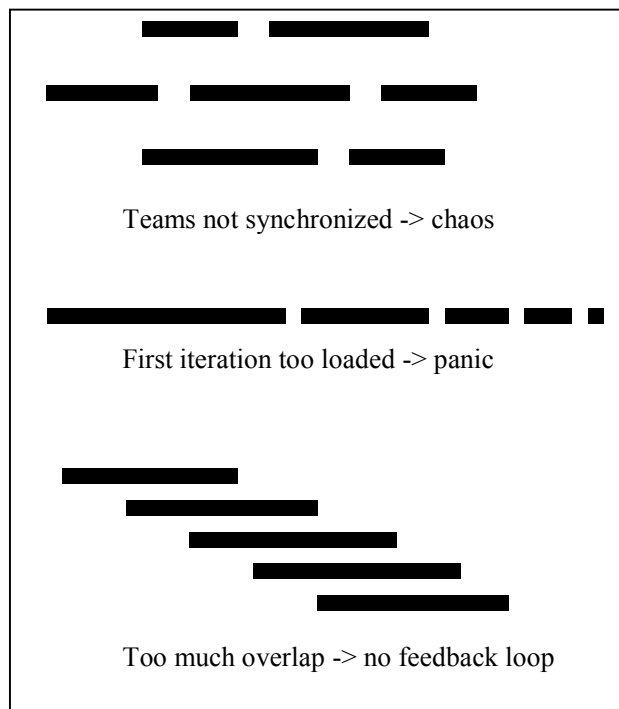


Figure 1: Some Dangerous Iteration Patterns

A good project manager and a good architect

To succeed, a software project needs both a good project manager and a good architect. The best management possible and iterative development will not lead to a successful product without a good architecture, and vice versa, a fantastic architecture will fail lamentably if the project is not well managed. It is therefore a matter of balance, and focusing solely on project management will not lead to success. The project manager cannot simply ignore architecture: it takes both architecture expertise and domain expertise to determine the 20% of things that should go into early iterations.

Trap: Use the same person as the PM and the architect

Using the same person as the project manager and the architect will work only on small projects (5-10 people). For larger endeavors, having the same person play the role of both project manager and architect will usually end with the project neither properly managed, nor well architected. First, the roles require different skill sets. Secondly, the roles, in and of themselves, are more than a full-time job. Therefore, the project manager and architect must coordinate daily, communicate with one another, and compromise. The roles are akin to that of a movie director and a movie producer, both work toward a common goal, but are responsible for totally different aspects of the undertaking. When the same person plays the same two roles, the project rarely succeeds.

Conclusion

At this stage, our reader may be discouraged: so many problems ahead, so many traps into which to fall. If it is so hard to plan and execute an iterative development, why bother? Rejoice, there are recipes and techniques to systematically address all these issues, and the payoffs are greater than the inconvenience in terms of achieving reliably higher quality software products. Some key themes: “Attack the risks actively or they will attack you.” (Tom Gilb) Software comes first. Acknowledge scrap and rework. Choose a project manager and an architect to work together. Exploit the benefits of iterative development.

The waterfall model made it easy on the manager and difficult for the engineering team. Iterative development is much more aligned with how software engineers work, but at some cost in management complexity. Given that most teams have a 5-to-1 (or higher) ratio of engineers to managers, this is a great tradeoff.

Although iterative development is harder than traditional approaches the first time you do it, there is a real long-term payoff. Once you get the hang of doing it well, you will find that you have become a much more capable manager, and you will find it easier and easier to manage larger, more complex projects. Once you can get an entire team to understand and think iteratively, the method scales are much better than traditional approaches.

Acknowledgements

Thank you to my Rational colleagues John Smith, Dean Leffingwell, Joe Marasco and Walker Royce for sharing your experience in iterative project management and helping me with this paper, which is now chapter 6 in our colleague Gerhard Versteegan's new book on software development, *Projektmanagement mit dem Rational Unified Proces*.

Further Reading

1. Walker Royce, *Software Project Management—A Unified Approach*, Addison Wesley Longman, 1999.
2. Tom Gilb, *Principles of Software Engineering Management*, Addison-Wesley, 1988.
3. *Rational Unified Process 5.5*, Rational Software, 1999.
4. Gerhard Versteegen, *Projektmanagement mit dem Rational Unified Process*, Springer-Verlag, Berlin, 2000.

Rational[®]

the software development company

Dual Headquarters:
Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Rd
Lexington, MA 02421
Tel: (781) 676-2400
Toll-free: (800) 728-1212
Tel: (408) 863-9900
Fax: (408) 863-4120

E-mail: info@rational.com
Web: www.rational.com
For International Locations: www.rational.com/worldwide

For International Offices: www.rational.com/worldwide

Rational, the Rational logo, Rational the e-development company, Rational Unified Process are trademarks of Rational Software Corporation. References to other companies and their products use trademarks owned by the respective companies and are for reference purposes only.

© Copyright 2000 by Rational Software Corporation.
Subject to change without notice. All rights reserved