This content is no longer being updated or maintained. The content is provided “as is.” Given the rapid evolution of technology, some content, steps, or illustrations may have changed.

Transitioning from waterfall to iterative development

Per Kroll

Published on April 16, 2004

Most software teams still use a *waterfall* process for development projects. Taking an extreme v
you complete a number of phases in a strictly ordered sequence: requirements analysis, design
and then testing. You also defer testing until the end of the project lifecycle, when problems ter
resolve; these problems can also pose serious threats to release deadlines and leave key team
periods of time.

In practice, most teams use a *modified waterfall* approach, breaking the project down into two
called phases or stages. This helps to simplify integration, get testers testing earlier, and provid
status. This approach also breaks up the code into manageable pieces and minimizes the integr
and drivers, required for testing. In addition, this approach allows you to prototype areas you d
feedback from each stage to modify your design. However, that runs counter to the thinking beh
Many design teams would view modifying the design after Stage 1 as a failure of their initial des
And although a modified waterfall approach does not preclude the use of feedback, it does not
encourage it. And finally, the desire to minimize risk does not typically drive a waterfall project.
improvements that an "iterative" approach to the software development process offers over the

Contents

Advantages of an iterative approach

In contrast, an iterative approach -- like the one embodied in IBM Rational Unified Process® or
incremental steps, or iterations. Each iteration includes some, or most, of the development disc
analysis, design, implementation, and so on), as you can see in Figure 1. Each iteration also has
objectives and produces a partial working implementation of the final system. And each succes
work of previous iterations to evolve and refine the system until the final product is complete.

Early iterations emphasize requirements as well as analysis and design; later iterations emphas
testing.

Figure 1: Iterative development with RUP. Each iteration includes analysis, design, implementation and testing activities. Also, eac the work of previous iterations to produce an executable that is c final product.

The iterative approach has proven itself superior to the waterfall approach for a number of reas

- **It accommodates changing requirements**. Changes in requirements and "feature creep" -- are technology- or customer-driven -- have always been primary sources of project trouble, dissatisfied customers, and frustrated developers. To address these problems, teams who u on producing and demonstrating executable software in the first few weeks, which forces a helps to pare them down to essentials.
- **Integration is not one "big bang" at the end of a project**. Leaving integration to the end aln consuming rework -- sometimes up to 40 percent of the total project effort. To avoid this, ea integrating building blocks; this happens progressively and continuously, minimizing later re
- **Early iterations expose risks**. An iterative approach helps the team mitigate risks in early it for all process components. As each iteration engages many aspects of the project -- tools, members' skills, and so on -- teams can quickly discover whether perceived risks are real ar not suspect, at a time when these problems are relatively easy and less costly to address.
- **Management can make tactical changes to the product**. Iterative development quickly pro architecture (albeit of limited functionality) that can be readily translated into a "lite" or "mo release to counter a competitor's move.
- **It facilitates reuse**. It is easier to identify common parts as you partially design or impleme recognize them during planning. Design reviews in early iterations allow architects to spot p reuse, and then develop and mature common code for these opportunities in subsequent it
- **You can find and correct defects over several iterations**. This results in a robust architectu application. You can detect flaws even in early iterations rather than during a massive testir can discover performance bottlenecks when you can still address them without destroying y panic on the eve of delivery.
- **It facilitates better use of project personnel**. Many organizations match their waterfall app organization: Analysts send the completed requirements to designers, who send a complete send components to integrators, who send a system for test to testers. These multiple hanc misunderstandings; they also make people feel less responsible for the final product. An ite wider scope of activities for team members, allowing them to play many roles. Project mana staff and eliminate risky handoffs.
- **Team members learn along the way**. Those working on iterative projects have many opport development lifecycle to learn from their mistakes and improve their skills from one iteratio each iteration, project managers can discover training opportunities for team members. In c waterfall projects are typically confined to narrow specialties and have only one shot at des
- **You can refine the development process along the way**. End-of-iteration assessments not project from a product or scheduling perspective; they also help managers analyze how to i and the process in the next iteration.

Contents

Some project managers resist adopting an iterative approach, seeing it as a form of endless, un RUP the entire project is tightly controlled. The number, duration, and objectives of iterations ar tasks and responsibilities of participants are well defined. In addition, objective measures of pr the team does rework some things from one iteration to the next, this work, too, is carefully cor

Four steps for a transition

Most waterfall projects divide the development work into phases or stages; we can also view th iterative design. But then, to move to an iterative approach, we would apply different process p four steps:

1. Build functional prototypes early.
2. Divide the detailed design, implementation and test phases into iterations.
3. Baseline an executable architecture early on.
4. Adopt an iterative and risk-driven management process.

Let's examine each of these steps more closely.

Contents

Step 1: Build functional prototypes early

As a first step toward iterative development, consider one or more functional prototypes during phases. The objectives of these prototypes are to mitigate key technical risks and clarify stakeh the system should do.

Start by identifying the top three technical risks and the top three functional areas in need of cl might relate to new technology, pending technology decisions that will greatly affect the overal requirements that you know will be hard to meet. Functional risks might relate to areas in which fuzzy requirements for critical functionality, or to several requirements that are core to the syst

For each of the key technical risks, outline what prototyping you need to do to mitigate the risks examples:

Technical risk: The project requires porting an existing application to run on top of IBM WebSpl are already complaining about the application's performance, and you are concerned that porti even more.

Prototype: Build an architectural prototype to try out different approaches for porting your app WebSphere architect to help you. Evaluate the results and write architectural and design guidel *dos* and *don'ts*. This will increase the likelihood that your ported application's performance will rework late in the project.

Technical risk: You are building a new application for scheduling and estimating software proje differentiator for this application versus off-the-shelf products will be how well it supports itera also one of the fuzziest areas in your requirement specification.

Prototype: Build a functional prototype based on your assumptions about how to support iterat demonstrating the prototype to various stakeholders, you will encourage them to pay more atte which of your assumptions they agree or disagree with. The prototype will help you clarify the p provide you with useful information about the user experience and look and feel for your applic reusable code.

Contents

Step 2: Divide the detailed design, implementation and test phas

Many project teams find it hard to divide a project into meaningful iterations before they know v But when you are ready to enter the detailed design phase, you typically have a good understar are, and what the architecture will look like. It's time to try out iterative development!

You can use two main approaches to determine what should be done in what iteration. Let's dis approach.

Approach 1: Develop one or more subsystems at a time. Let's assume that you have nine subsystem numbers of components. You can divide the detailed design, implementation and test phase into thre implementing three of the nine subsystems. This will work reasonably well if there are limited depend subsystems. For example, if your nine subsystems each provided a well-defined set of capabilities to the highest priority subsystems in the first iteration, the second most important subsystems in the se approach has a great advantage: If you run out of time, you can still deliver a partial system with the r and running.

However, this approach does not work well if you have a layered architecture, with subsystems in the capabilities of subsystems in the lower layers. If you had to build one subsystem at a time, such an ar build the bottom layer subsystems first, and then go higher and higher up. But to build the right capab typically need to do a fair amount of detailed design and implementation work on the upper layers, be need in the lower layers. This creates a "catch-22"; the second approach explains how to resolve it.

Approach 2: Develop the most critical scenarios first. If you use Approach 1, you develop one subsy you focus instead on key scenarios, or key ways of using the system, and then add more of the less es different from Approach 1? Let's look at an example.

Suppose you are building a new application that will provide users the ability to manage defects. It is of WebSphere Application Server, with DB2 as the underlying database. In the first iteration, you deve as entering a simple defect, with no underlying state engine. In the second iteration, you add comple example, you might enable the defect to handle a workflow. In the third iteration, you complete the d providing full support for atypical user entries, such as capability to save a partial defect entry and th

With this approach, you work on *all* the subsystems in *all* iterations, but you still focus in the first itera and save what is least important or least difficult for the last iteration.

Approach 1 is more appropriate if you are working on a system with a well-defined architecture existing application or developing a new application with a simple architecture, for example. Mo applications should use Approach 2, but they should plan the iterations in such a way that they iterations to make up for possible schedule delays.

Contents

Step 3: Baseline an executable architecture early on.

You can view this step as a much more formal and organized way of doing Step 1: *Build functior* what is an "executable architecture"?

An executable architecture is a partial implementation of the system, built to demonstrate that support the key functionality. Even more important, it demonstrates that the design will meet re throughput, capacity, reliability, scalability, and other "-ilities." Establishing an executable archi the system's functional capability on a solid foundation during later phases, without fear of brea architecture is an *evolutionary prototype*, intended to retain proven features and those with a hi system requirements when the architecture is mature. In other words, these features will be pa contrast to the *functional prototype* you would typically build in step 1, the evolutionary prototy architectural issues.

Producing an evolutionary prototype means that you design, implement, and test a skeleton str system. The functionality at the application level will not be complete, but as most interfaces b implemented, you can (and should) compile and test the architecture to some extent. Conduct tests. This prototype also reflects your critical design decisions, including choices about techno their interfaces; it is built after you have assessed buy versus build options and after you have c architectural mechanisms and patterns.

But how do you come up with the architecture for this evolutionary prototype? The key is to foc 30 percent of use cases (complete services the system offers to the end users). Here are some cases are most important.

- **The functionality is the core of the application, or it exercises key interfaces**. The system' determine the architecture. Typically an architect identifies the most important use cases b redundancy management strategies, resource contention risks, performance risks, data sec example, in a point-of-sale (POS) system, Check Out and Pay would be a key use case becau a credit card validation system -- and it is critical from a performance and load perspective.
- **Choose use cases describing functionality that *must* be delivered**. Delivering an applicatio would be fruitless. For example, an order-entry system would be unacceptable if it did not a Typically, domain and subject-matter experts understand the key functionality required fron behaviors, peak data transaction, critical control transactions, etc.), and they help define cri
- **Choose use cases describing functionality for an area of the architecture not covered by** ensure that your team will address all major technical risks, they must understand each are certain area of the architecture does not appear to be high risk, it may conceal technical diff only by designing, implementing, and testing some of the functionality within that area.

The first and last criteria in the above list will be of greater concern to the architect; project mar first two.

For each critical use case, identify the most important scenario(s) and use them to create the e words, design, implement and *test* those scenarios.

Contents

Step 4: Adopt an iterative and risk-driven management process.

If you were to follow Steps 2 and 3 as described above, then you would come very close to the development. Then, your next step would be to fuse Steps 2 and 3, adding a management lifecy and iterative development. That is the iterative lifecycle described in RUP.

RUP provides a structured approach to iterative development, dividing a project into four phase Construction, and Transition. Each phase contains one or more iterations, which focus on produ necessary to achieve the business objectives of that phase. Teams go through as many iteration objectives of that phase, but *no more*. If they do not succeed in addressing those objectives wit they had planned, they must add another iteration to the phase -- and delay the project. To avo on what you need to achieve the business objectives for each phase. For example, focusing too Inception would be counterproductive. Below is a brief description of typical phase objectives.

- **Inception**: Establish a good understanding of what system to build by getting a high-level u requirements and establishing the scope of the system. Mitigate many of the business risks building the system, and get buy-in from all stakeholders on whether or not to proceed with
- **Elaboration**: Take care of many of the most technically difficult tasks: design, implement, te architecture, including subsystems, their interfaces, key components, and architectural me with inter-process communication or persistency). Address major technical risks, such as re performance risks, and data security risks, by implementing and validating actual code.
- **Construction**: Do a majority of the implementation as you move from an executable archite version of your system. Deploy several internal and alpha releases to ensure that the systen users' needs. End the phase by deploying a fully functional beta version of the system, inclu documentation, and training material; keep in mind, however, that the functionality, perform system will likely require tuning.
- **Transition**: Ensure that the software addresses the needs of its users. This includes testing release and making minor adjustments based on user feedback. At this point in the lifecycle mainly on fine-tuning the product, and on configuration, installation, and usability issues; al should have been worked out earlier in the project lifecycle.[1]

Contents

Many ways to apply these steps

In this article, we have described how you can gradually transfer from a waterfall approach to a approach, using four transitional steps. Each step will add tangible value to your development e Some teams may take on more than one step at a time; others may run a few projects based on next step. However you choose to use this step-wise approach, it can help you reduce the risks changes in a development organization.

Notes

1 For a detailed description of what a RUP lifecycle looks like in practice, see Chapters 5-8 in *Th Made Easy*, by Per Kroll and Philippe Kruchten (Addison-Wesley, 2003).

Downloadable resources

PDF of this content

Comments

Sign in or register to add and subscribe to comments.

- [ ] Subscribe me to comment notifications

Tutorials

Recipes

Open Source Projects

Videos

Newsletters

Events

Cities

Developer Answers

Contact Privacy Terms of use Accessibility Feedback Cookie Preferences

Unit