Managing the Iterative/Incremental Development Process

By Timothy D. Korson

There is widespread agreement that an iterative/incremental development process is the best process to follow when using OO development techniques. There is also a fairly good high-level understanding of what it means for developers to follow an opportunistic iterative/incremental development process. What is typically not understood, is how the interactions between iterating over project deliverables and incrementally completing the system work out so that the project moves forward in a disciplined way that managers can plan and track. This column outlines these interactions. I will start by defining terms and then give an overview of the process. The rest of the column will elaborate on the process. There are many variations on the iterative/incremental development process. Many are less disciplined then what is described in this column. The focus of this column is a **manageable** iterative/incremental development process.

DEFINITIONS

An increment is some defined subset of the system. A completed increment is some defined subset of the system for which the analysis, design and code of the system is completed, reviewed and tested. Increments can be internal or external. An external increment is one for which some subset of the ``use cases'' is implemented and therefore the system has some externally visible functionality. An internal increment cannot be traced to a specific set of use cases. An internal increment typically corresponds to a completed subsystem such as database access, inter-process communication or other system facility.

An iteration is a pass over some set of activities. An iterative process is when I plan to revisit previous work. The iterative philosophy is: analyze a little, design a little, code a little, test a little, learn a lot. I then take this learning and use it to rework previous deliverables as I iterate over the development phases.

A prototype is something I build to verify or validate a concept, to assess feasibility, gather performance data, get feedback on user interface issues, etc. The purpose for building a prototype is to gain information. Once the prototype has yielded that information, the prototype doesn't owe the project anything more.

A manageable iterative/incremental development process combines increments, iteration, and prototypes in the following manner. The overall macro process is incremental. Iteration occurs within increments and across unfinished deliverables, with heavy prototype support as necessary and useful. Often this will involve reworking one piece of the system several times before an increment is finished. Previous increments are only revisited to fix errors or serious flaws. To get started, do enough analysis and design to form a context, then concentrate on driving an increment through to completion (running, tested code).

DETAILS

To get started, do enough analysis and design to form a context. This step involves creating the first iteration of the domain models[2] for a significant portion of the core business domain, gathering the top level application use cases, and doing a first iteration of the overall design. The time taken in this start-up ``waterfall'' step should not exceed 2 months even for very large projects. For a small project it could take less than 2 weeks. This first quick ``waterfall'' pass is taken for several reasons. It yields the information needed for scoping the project and making the first cut on a more detailed project plan. It also forms a context, and outlines project constraints, for the analysis and design of each of the increments.

The overall macro process is incremental. The management task here is to determine what functionality should go into each increment, how the increments should be sequenced, and what resources to allocate to the increment. To accomplish this task a manager must balance political and technical risks against project productivity. The political risk of project cancellation is minimized by scheduling key external increments early. Other factors being equal, projects are less likely to get canceled once part of the code is up and running and the customer can see the system execute. Technical risks are minimized by scheduling technically risky increments early, lots of early prototyping, and early scheduling of external increments that exercise large portions of the design. Project productivity and time to completion (if all goes well) are maximized, however, by doing low risk, internal increments first. This is because completing an internal increment with a known design solution involves simply implementing all the methods of the small set of classes that form that ``sub-system.'' Classes can be assigned to developers, team communication needs are minimized, so productivity is high. Knowledge gained while working these low risk increments can give the team insight into less well understood increments that they will be subsequently developing.

Completing an external increment, however, usually involves coding some portion of the methods of a large percentage of all the classes in the system. Team communication needs are high, and the same classes will need to be touched again in a later increment so productivity goes down.

To intelligently select and schedule increments, a project manager will typically need to interact closely with the project technical leads. The first draft of the project plan will include scheduling information for, and the contents of, the major project increments. As the project proceeds, the project plan will be revised, and more detail will be added.

Iteration occurs within increments. To complete an increment, a project team will iterate multiple times over the analysis, design, implementation, integration, and testing of the classes in this increment. The first pass may be a quick setup pass where issues are scoped, risks are assessed, and resources are obtained and allocated. Next there may be one or more prototyping passes where issues are explored and solutions are refined. Then one or more development iterations occur during which the production classes are implemented. Finally there is a clean-up iteration where deliverables are reviewed, code is polished, and documentation is finished for that increment. This increment is now considered complete. One does not return to any of the deliverables in this increment unless an error or serious flaw is discovered, or a requirements change that affects this increment is negotiated. Completion of an increment is a major project milestone.

Iteration occurs across unfinished deliverables. Once increment N is complete, the project plan is revisited and final determination is made of the contents and schedule for the next increment which we will call increment N+1 (the next increment will actually be N+a if there are ``a'' teams working in parallel on the project). As the team works on increment N+1 they will iterate over the project analysis and design models. At this point, the portions of the models for increments one through N will be complete and will not be revisited for purposes of incremental improvement, but the portions of the models for the remaining increments will be reexamined and updated to reflect the most current understanding of the problem domain and solution. In addition, detail relevant to increment N+1 will be added to the models, and the corresponding detail design and implementation will be completed.

Notice the interaction and balance between iterative and incremental. Iterative drives one to get it right. Incremental drives one to get it done. Iterate within an increment until that increment is right. Once an increment is finished, don't revisit it, only iterate over the pieces that are not yet a part of a finished increment. Even when iterating over the pieces of the analysis and design not yet locked into an increment, focus on driving the current increment to completion. Unfinished parts of the model are revisited only to the extent necessary to keep them consistent with completed increments and to keep the current increment consistent with overall project constraints and architectural decisions.

Heavy prototype support as necessary and useful. Engineering firms design carefully and create lots of prototypes before moving to production. Prototypes reduce risk by providing low cost answers to questions that arise during the project. These questions could be about the specifications, the architecture, performance, memory constraints, or any other project consideration. This leads to the notion of analysis prototypes, design prototypes, and implementation prototypes. This topic is complex enough that it could become a whole separate column: ``Managing prototyping in your process.''

Previous increments are only revisited to fix errors or serious flaws. Sometimes, even in the most carefully managed projects, serious errors or design inconsistencies are uncovered in an already completed increment. I don't consider this a part of the standard iterative process.

When this happens the project schedule is invalidated and the project may incur cost or schedule overruns. Given the flexibility of object architectures, a project can sometimes recover without cost or time overruns, but usually this scenario entails a visit to the customer with bad news. In my experience, good management can usually prevent this from happening by careful planning of the contents and sequencing of the project increments coupled with adequate prototyping and sufficient iteration within increments.

Written: September 1996
Current Website: http://qualsys-solutions.com