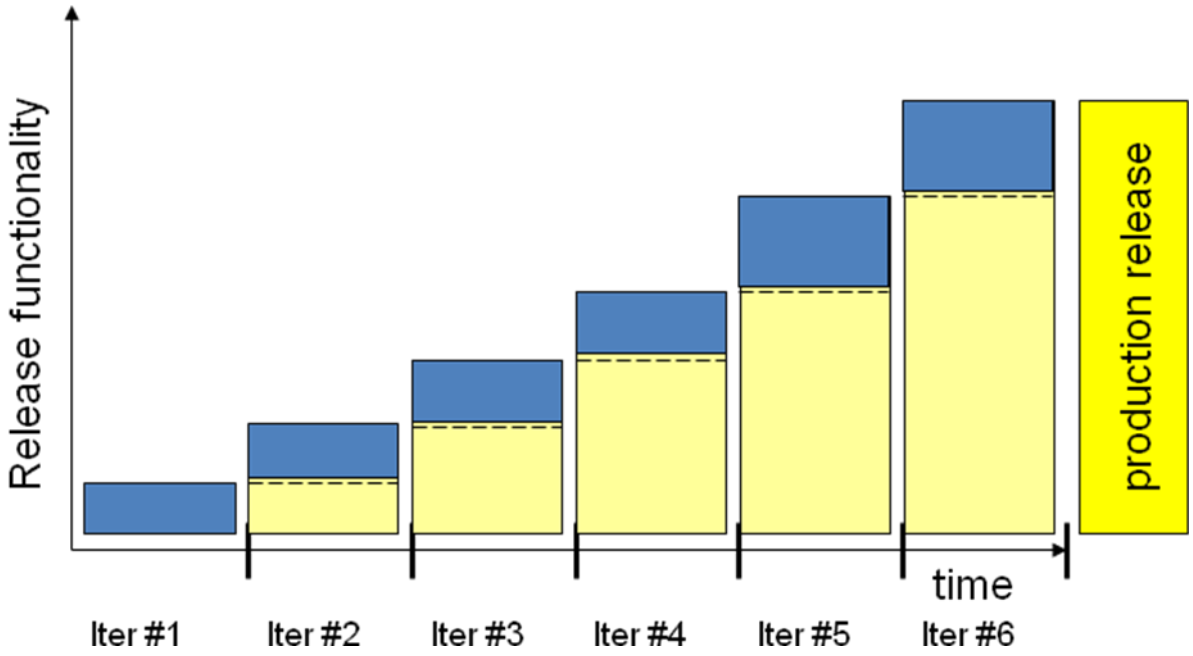
RDF Group Best Practices: Incremental and Iterative Development

In my 10 plus years working at RDF Group, our practices have greatly evolved, largely due to lessons learnt along the way – some of which I would like to share with you.

Key Principle #1: Incremental and Iterative Development

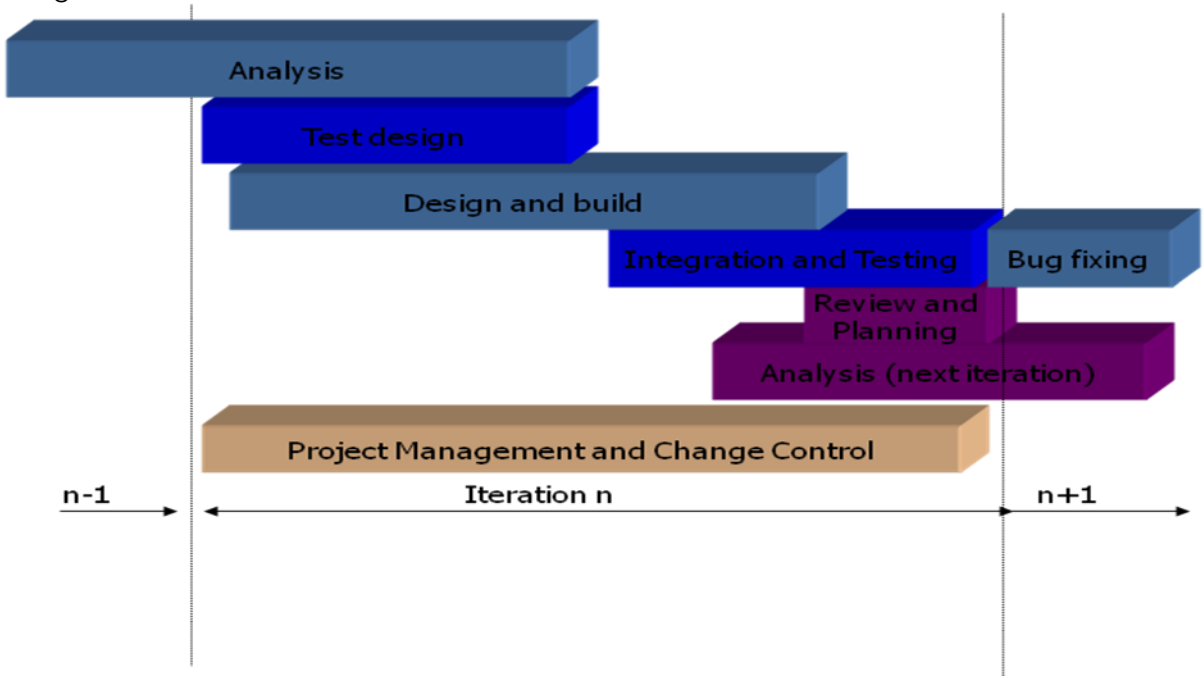
'Iterative' means that the project is planned as a sequence of time-boxed development cycles, or iterations.
'Incremental' means that the solution is built, tested and in some cases delivered in increments. Each iteration produces a release (aka increment), incrementing the functionality of the release built in the previous iteration. The picture below shows how the solution grows over a hypothetical project with six iterations.



The advantages of iterative and incremental development over a one-shot approach are well known, including reduced risk of failure and rework, better adaptability to changing requirements, and many others. This style of development has served us well. I believe it has been a vital factor in ensuring the success of our projects and consequent retention of our key clients.
Iterative and Incremental Development (IID) has been known to the Software Engineering community for a long time – see the 2003 article by Larman and Basili. However it became widely popular in the 1990s, mainly through the Rational Unified Process (RUP), a method published by Rational, where the three OO amigos, Grady Booch, Ivar Jacobson and James Rumbaugh had converged.
RUP is not a rigid methodology, rather a framework that can be adapted to the needs of each project. At RDF we chose RUP as a framework to support our newly formed OO team in the year 2000, and we adapted it over the years to the needs of our projects.
Here are some reflections on how we have used incremental and iterative development at RDF.

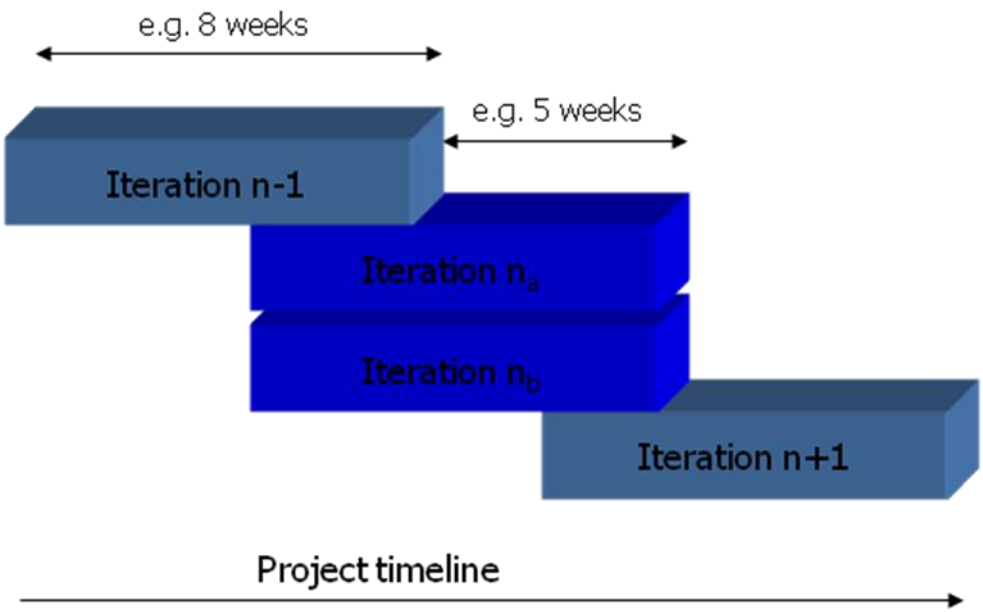
Overlapping iterations

Work must flow smoothly amongst the different workstreams (analysis, design and build, testing), keeping all team members fully engaged on the project the whole time. This is important to ensure maximum productivity, avoiding dead times, and to ensure all roles can be charged to the client for the full duration of the project, while providing maximum value from their time. A developer should never be waiting for the analyst to complete a specification, for example.
We quickly found that one effective way of achieving this was to start work on defining/specifying the functionality of the next iteration while the current iteration is still in development and testing. The following image gives a rough idea of how this can be achieved. The analysis stream is responsible for client facing activities to capture detailed requirements, build prototypes and obtain client sign off. This work is done in each iteration and needs to be, if not completed, at least in a stable outline stage in order to proceed with design and build.



Iteration Length

The length of the iterations depends on a number of factors, including the size and complexity of the entire project and the experience and velocity of the team. A length of six to eight weeks for complex projects lasting six months or more is common. Smaller less critical projects can have two or three week long iterations. The longer iterations are needed for the large and/or complex projects, because there is just more functionality to produce, more problems to solve, more analysis and design to take place, and more extensive testing.
Increasing the team size to reduce iteration length works up to a point. Teams must be small to work in an agile fashion, with frequent workshops and reviews. A typical team size is five to eight. We have found better to have multiple teams working to the same project plan than to try and increase the team size beyond eight to try and decrease iteration time.
With overlapping iterations, as described above, iterations are longer than the project heartbeat, i.e. the time between one release and the next. For example, with a 3 week overlap, the iteration length can be eight weeks, while the project heartbeat is 5 weeks – see below.



Time boxing means iterations cannot be extended

An eminent software practitioner was once asked how a project can deliver one year late. He answered "One week at a time". We know by experience that postponing the completion of an iteration, to accommodate new requirements or because the effort was under-estimated and planned functionality is incomplete, is a sure way of making the entire project late. Time lost cannot be recouped in subsequent iterations, as the project heartbeat cannot be easily modified. It is far better to leave some requirements/ functionality out, re-planning the contents of the coming iterations, than to prolong the current iteration. Doing so disrupts the activities of the workstreams, slowing down the project even further. What is taken as a measure to catch up backfires badly. The "Review and Planning" activity in each iteration is the point at which we must fend off requests, often by the client, to postpone delivery and instead re-plan subsequent iterations to take into account any new events/dependencies/delays.

In the next post, I will move onto another principle, use case driven development, and discuss the RDF way in relation to use cases.