The Iterate and Increment Model in the MMIXCode Project

Written by James Yoder

For the MMIXCode, I ended up using the Iterate and Increment (I&I) model. Surprisingly, it made the process simpler and the design more straightforward. Several times the design changed or the functionality was scaled back and the I&I model adapted to the changes while keeping the design orderly and uncomplicated.

The requirements was simple since I already had a view of what the program ultimately would do, but for major projects, the requirements phase nails down the overall goal and gives a birdseye view of the project from the beginning. For clients, the requirements phase lets the developers know exactly what they expect from the program. A prioritized list helps to keep the needed functionality first if the scope must be scaled back.

Unlike the requirements phase, the analysis phase often is composed of several different documents and contains the full external interface from different perspectives. The functional specification contains details about how each function of the program acts. The user interface specification is concerned about how the application looks to the user. The architectural specification describes the different external components and how they fit together. In MMIXCode, all the specification documents were combined into a single document that was divided into sections. This document was changed several times as features were scaled back, the primary advantage of the I&I model over the waterfall model.

Once the functionality is known, the next step is to visualize the internals of each component and split them up into classes. Classes should represent logical objects in the component and have the needed properties and methods to perform their jobs. Private members were omitted since the design
specified the interface of the classes and the internals were done in the implementation step. Usually, each component has its own design specification, but MMIXCode is small enough that I put them all into a single document that was divided into sections for each component. This document was often updated as the program was implemented and class needs changed.

After the big plan was down on paper, the implementation was straightforward. The written directions were turned into C# code and fitted together into a complete program. The internal members and algorithms of each class had to produce the functionality described by the design specifications. Often, the designs needed to change to accommodate unforeseen problems. Implementation in the I&I model is much speedier than with the code and fix model because delays deciding class layout are eliminated and only small decisions about internals and algorithms need to be made. The implementation of MMIXCode went smoothly except for several instances where the designs needed to be modified.

In conclusion, the incremental and iterative model worked well for my small project that did not have any major requirements changes. Having design documents helped to visualize the project and decide beforehand how the entire system would work together.