IN THE NAME OF THE PROCESS

by Kris Oosting

We do not want process to be overshadowed in all the excitement around the emergence of notation standards for OO analysis and design. This article explains why having a process is so important.

During the past three years we have extensively observed and measured Fusion projects all over the world. We discovered that projects where developers only used parts of the Fusion notation, but did not follow its process, had problems in the area of consistency, communication between developers, quality of the software (higher complexity, more defects), and controlling the project. Ignore the process and project risks amass; follow the process and you have a guide to success.

Which is Faster?

The question always is: "Which is faster—starting with a method or starting with coding?"

When starting with coding, there is one "advantage." After a very short period of time a demonstration can be given. You can see that something is happening. And something is happening indeed! After the initial creation phase, the fun starts. So much code was created so quickly that we lost track. The rest of the time we will rework code, do bug-fixing, get irritated, have no time for documentation, have no time to do proper testing. There is no process in place, "because we are artists, you know." But, the outside world can still see that something is happening. Actually, what they see is fire fighting—which is still rewarded quite well in some companies, because we're working very hard and others can see it.

On the other hand, if we use a method and follow its process, we will not start coding directly. For most people this means that "nothing" happens. We cannot give a demo after doing a few weeks' analysis and some design. We can show beautiful pictures called designs, but no real working parts of the system yet. We *do* know, however, exactly what the problem is and how to solve it, but nobody seems to be interested.

From our experience and measurements, we have concluded that most management, like steering committees, start to panic after about 2 to 3 months. They will try to force the team into coding (hacking) because they want to see something. With this action they demonstrate again that developers are paid to hack!

Example from the Real World

In one case we had an exception where management went for about 10 months before starting to panic. Then the decision was clear: "Start coding so we can demo the application. When this is done we'll give you the time to finish off the design." The basic framework for the application was finished, but this was not very exciting to demo. So coding started. For 2 weeks people spent, as a team, 80% of their time coding. The demo was a success for outsiders, but not for the developers—they spent 3 weeks updating the designs and rewriting code to follow the designs. They did this not because they had to, but because they wanted to.

In this case management was lucky. The developers had spent a lot of time designing already. Formally it was not finished, but they knew what to do. The Object Interaction Graphs received a lot of the credit. The developers also reported that they had a guideline, a "prime directive," to rely on—this was the process, telling them what to do when, and how to measure the results.

Another nice thing is that after one year of running this project, the schedule over-run only is around 15%. Before, with their OMT/Coad-Yourdon projects it was around 200%. The process of the Fusion method gives them a framework to fit the notations into. As a result people do not wander around without a goal or a direction. They have a guideline to follow and know why they are doing it. Creating a model using a notation suddenly has a meaning. It is part of the whole plan, not just a little diagram with no connection to other diagrams.

Other real-life examples

We came across another example of the value of having a process. A company we know of wanted a little application to be built. A software house estimated the price for building the application at 140K. The internal department, which has a process in place, estimated a conservative cost of 40K. They did it for just over 10K.

If you understand your process, you can estimate better. If there is no process, or an ill-defined one, estimation is difficult and a lot of money can be wasted unnecessarily.

Another advantage of having a good method with a process in place is that documentation is created *during* development. There is no hurried we-have-to-create-documentation phase after implementation!

We want to develop better software faster with less maintenance effort. Time to Market is vital. Though we want better products faster, we do not really want to "invest" in it. Investing means to learn, to change, to adapt. Not only to buy tools, but to have a process—you probably already know that a tool on its own does not provide you with a good process.

Also having the right process in place for the right problem to solve is important, as a big oil company discovered. They used a function-point expert to estimate the effort and price for developing a new application. Although the expert followed a well-defined and tested process, he was still 1200% off! Object-oriented system development behaves differently than traditional system development.

More than just a Notation

What we need is more than just a collection of notations. Some methods are a collection of notations, others are a collection of notations with a supporting process.

The Unified Method, now called the Unified Modeling Language (UML), is a collection of notations. So it is only fair that they changed its name to UML. This also indicates that methods which do not have a good process are actually forms of modeling languages, or a kind of palette for a drawing tool.

This does not take away the fact that it is a good idea to have a unified notation for modeling certain aspects of application development. We just hope that it will not become too difficult to draw and understand.

The Fusion method was a good step in the right direction. The next generation of Fusion will take us even further. Supporting requirements, analysis, design, coding, planning, group development, and much more. One thing they did not forget—the process. It's more than just a notation!

Just enough Process

The difficult bit is not to overdo it. This means not too much process. As W. Humphrey told us before: "the true cause of errors is the process, not the people." "Well then," you may say, "skip the process, and the errors will be gone." Similarly, the TRW studies showed 42% of all defects have their cause in design. So skip design and 42% of the defects are gone. If this is our argument, we have just forgotten Murphy's Law!

The game is not to omit the process, but to have *just enough*. The same goes for the method. Have just enough method and not a huge monster that can tackle every problem—if you're lucky enough to be able to handle the beast!

One of the Fusion team's strong points is that they constructed a method that is just enough. It is a framework that is ready to be used. Just as an application framework is more than an object library, a method framework is more than just a collection of notations.

A process is transforming one model to another. It also supports the fact that models can be used as checks against each other. And last but not least, a process makes a method measurable and controllable.

A few metrics can help us to tell if the process is working or not. We have had good results with the following metrics:

- *Estimating Quality Factor* (EQF)—the quality of the estimates (time and $$)
- *Defects*—Defect Tracking analyzes the process
- *Time*—progress (see also Fusion Newsletter of July 1996, page 12—14)

If you know what is going on, and why, you can do something about it.

Describing a process does not need to take up 2 meters of bookshelf. It can be done in a matrix or table format, especially when people are not experienced with methods like Fusion. The following items can be a part of such a process description:

- *Phase*—for example Design
- *What to model*—the model you have to make, e.g. OIG
- *Needs as input*—what you need to transform, information to construct the model from
- *Needs to produce*—the deliverable
- *Process*—sort of “business rules” describing the transformation
- *What to measure*—how do we know it works well
- *What tools to use*—to use the right tool for the right job

Conclusion

We have observed and audited many projects, mainly object-oriented, Fusion and non-Fusion. Success or failure often came down to one thing—the process.

Developers know how to draw diagrams. But how these diagrams support and check each other was often not clear. This was due to the fact that the process did not exist, partly or completely, or was ill-defined. Therefore it was not measurable. So, with nobody in control—well, you probably know the rest of the story...

May the process be with you!