ANALYSIS IN THE REAL WORLD

by Kris Oosting,

This article will briefly describe some experiences with the Analysis phase of Fusion for an OSF/Motif C++ project at one of our customers. The application uses a GUI, has to work with 2D and 3D images, stores objects in an object database, and uses a complex calculation engine to produce the additional data for the images. The project size is about 12 - 15 person- years. The team that did the analysis consisted mainly of inexperienced Fusion users, though they had a background in OMT and Coad/Yourdon.

The questions we had were:

- what are the important models?
- how do (did) we use the method?
- what metrics can we collect?
- how can we test the models?

I will now describe some of our experiences.

What are the important models?

Well, that was easy to decide. The system object model and the operation model. But... later we discovered that the life-cycle model was necessary for our test as well, so it became more valuable again.

Developers often think it a waste of time to model their software. But just ask them how much functionality they have to implement—most of the time they cannot answer that. We learned that the number of system operations is an indicator for the amount of functionality and the effort to design it.

In developing the object model you need to

understand what the "lego blocks" are to play with and how you can stick them together. And if you want to "play" with the model, i.e., simulate the system, you'll need the life-cycle model.

So for this project the system object model, operation model, and life-cycle model were the most important ones and there is no resistance to creating them, after we did the simulation.

Timeline diagrams were not appropriate here. The developers found it too much work and the diagrams did not add significantly to the existing models.

How do (did) we use the method?

The following percentages give an indication of how the Analysis time was used:

- 7% for finding the "Lego blocks," i.e., reading the Requirements document and looking for classes, relationships, etc.;
- 16% for updating the GUI (the main GUI was created during the requirements phase);
- 29% for constructing and documenting the object models;
- 12% for making all the life-cycles;
- 36% for describing the system operations in operation models.

The developers found that describing the system operations was the most difficult part, but it had a positive side-effect—it formed a kind of review of the object model.

The team also mapped the dialogs (windows) of the GUI onto relationships between an agent and a class.

For example, *User enters Customer*, where *User* is the agent, *Customer* a class and *enters* the relationship. If there is no checking on the text fields and the *enters* consists of entering data in text fields and then selecting the OK button (and of course a CANCEL button to cancel it), then there is only one (or two) system operation called *enterCustomer* (and *cancelCustomerEntry*). If something is wrong, the system can respond by showing an Alert Dialog on the screen, which is an Output event. The life-cycle expression is:
(enterCustomer . [#customerEntryErrorDialog] | cancelCustomerEntry)

For example, 3 relationships, which produce 5 system operations and 2 output events, were represented by one Dialog.

This made testing, i.e., simulating the system, much easier and more effective.

The design uses a "Controller Object" to communicate with the GUI and the Customer objects. This was done to make sure that not every customer object has its own dialog, and if the GUI will change (porting software) only the controller objects will change, not the model (enterprise) objects.

During the analysis phase all models were used except for the timeline diagrams. The process of constructing the models was more or less the same as described in the Fusion book (Coleman et al., 1994). This was probably because we were dealing with beginning Fusion users. The developers did not find creation of the models difficult. However, they found that in doing the Operation Models it was very important that the GUI and System Object Model were clear and well described.

What metrics can we collect?

To understand our models and development process, we started to collect metrics.

We focused on just a few simple ones like:

- number of classes;
- number of attributes;
- total number of relationships;
- number of relationships between agents and classes;
- number of system operations;
- number of output events;
- number of relationships in System Object Model;
- mean number of attributes per class (derived);
- mean number of system operations per relationship on boundary (derived);
- mean number of output events per relationship on boundary (derived).

The metrics are collected after the models are updated following a formal review or test. During the review defects are registered as well. This is done to get an idea of how certain defects influence the models. Table 1 gives an idea of the values of the collected metrics.

Table 1: Analysis phase metrics

Metric	Value
number of classes	35
number of attributes	87
total number of relationships	99
number of relationships between agents and classes	60

Table 1: Analysis phase metrics

Metric	Value
number of system operations	191
number of output events	97
number of relationships in System Object Model	39
mean number of attributes per class	2.5
mean number of system operations per relationship on boundary	3.2
mean number of output events per relationship on boundary	1.6

These metrics indicate that the system is user interface intensive (number of relationships between agents and classes, number of system operations). All attributes are probably not entered since the mean number of attributes per class is low. The relationships on the boundary (between agents and classes) are probably representative of what is happening, and the mean number of system operations per relationship on the boundary is not too high.

We collected these metrics over time. In one example we found that the number of classes and attributes increased, but the number of relationships did not. This indicates that classes are aggregated—maybe even replacing attributes.

In another example, the number of system operations and output events also did not change during the last three measurements. This means that even though the object model changed, no new functionality was added to the system. This was true because the number of relationships between agents and classes also did not change. The number of relationships between classes did increase. To us this meant that although the functionality did not change, the model that must support it did.

CASE tools can easily provide these metrics.

In the following Fusion Newsletter I hope to present the connection with design metrics and if the assumptions made by using the above metrics were correct.

How can we test the models?

The question we had was: “how can we test the models produced in this phase (analysis)?”

We defined the Fusion Test Model (FTM) in order to guide the testing. The Fusion Test Model has been created by Shared Objectives and is reaching it final stage (i.e., it passes the real world tests).

The FTM is used to minimize the number of defects and to optimize the quality of the application produced. The FTM is part of the QA Handbook or Software Development Handbook and describes the following:

- Name of the test;
- Purpose of the test. Motivation to perform the test;
- Test method to use. For example, Walkthrough, Inspection, etc.;
- Models involved during test. For example, System Object Model and Life-cycle Model;
- When to perform the test in the software life-cycle;
- What to test. Describes what subjects/items are important to test;
- Metrics that can assist to get an idea of what is going on;
- Comments to support free text.

At the time of submission of this article we were just about to start with the Scenario Test.

The purpose of the Scenario Test is to test the scenarios as described in the user requirements against the user interface, i.e., how well does the system (object models, operation models, life-cycles) support the scenarios the user described. A scenario is a single function or collection of functionality to be supported by the system. A scenario also indicates a certain sequence of working.

The scenario test is actually a “simulation” of the system.

I will keep you all informed about our findings.

Other tests that will be performed are:

- **Object Interaction Test**. To test if the objects in the object interaction graphs communicate in the right way to support the requested functionality;
- **Object Reference Test**. To test the reference structure between objects involved in design;
- **Class Inheritance Test**. To test the inheritance structure to check for optimal generalization and specialization;
- **Method Test**. To test the implemented methods to determine that the logic, the computation, and data handling is correct;
- **Object Test**. To test the interface of the coded objects, after the methods are tested;
- **Module Test** (collection of objects). To test a group of objects, supporting a defined group of functionality;

- **Application Test**. To test the complete working application;
- **Requirements Tracking Test**. To test if the requirements can be tracked down from the user specification to code and back.

A follow-up article will describe more about the design tests including metrics collected.

Kris Oosting