# The Fusion Process from an Object-Oriented Perspective

*Pavel Hruby*
*Navision Software a/s*
*ph@navision.com*

Many software developers use the Fusion method with some modifications and adaptations to meet their specific needs. We at Navision Software a/s began to use the Fusion method about a year ago. We supplemented the method by working with use cases and incremental development. We also defined a relationship between Fusion and the project management process and made several other minor changes in order to be able to use UML as a notation.

In our modified definition of Fusion, we adopted several object-oriented features. Deliverables were considered objects, with attributes and methods. Evolution during software development was represented by object interactions.

This article describes a framework that focuses on flexibility, which makes adaptation of the method easy. We will discuss an object-oriented approach to the development process based on Fusion and our experience with it.

## *Basic Features of an Object-Oriented Process*

1. Deliverables produced during software development are considered objects with various attributes and methods.
2. There is a strict distinction between the *deliverable* and its *representation*. We use the term *model* for the deliverable and *diagram* for its representation if it can be represented graphically. The *model* determines what information we work with, and the *diagram* determines how it is represented. A class model, for example, is typically represented by one or more class diagrams. A use-case model, however, can be represented by a use- case diagram, a list of use-cases (text), use-case schemes, text describing sample scenarios, and so on. A system interaction model can be represented by one or more sequence diagrams, collaboration diagrams, state diagrams, in Backus-Naur form (Fusion life cycle), and so on. For each class of deliverable, there is a recommended type of representation.
3. Each increment (a small piece of functionality added to the existing product) is defined by a single deliverable called a *task context* document, which corresponds to a *task* in Microsoft Project. The task context document contains general information about the increment, such as a synopsis, requirements, metrics (time estimates), responsible developers, and so on . See Figure 1. for a complete list of attributes. The task context document exists throughout the entire life cycle of the increment (from the requirements analysis to implementation and testing). The development phase of the increment is represented as a value of one of the attributes of the task context document.

## *Classes of Deliverables*

Classes of deliverables determine behavior, semantics, and relationships of concrete deliverable instances (objects). Classes of deliverables have:

- Constructors, which are methods describing how to create a deliverable
- Quality-assurance methods, such as completeness and consistency checks

Examples of two deliverable classes, the abstract deliverable and the task context are shown in Figure 1.

Deliverables have numerous attributes:

- Kind, which determines deliverable class
- Name
- Description, typically UML diagram or text
- References to other deliverables, for example a use-case model is related to one or more system interaction models
- Project
- Subsystem
- Task context
- Responsible developer
- Other attributes, such as who created and modified the deliverable and when

Kind and name together are the key that uniquely identifies the deliverable in the data dictionary. Concrete deliverable classes specify methods and can include other specific attributes. An inheritance diagram of deliverable classes is illustrated in Figure 2. Typical relationships between deliverables are illustrated in Figure 4. The development process is illustrated in Figure 3.

Note that design deliverables may relate to more than one task context document. For example, different developers or teams may work with the same class model in the context of different increments or, more

typically, the same class model may be reused within different increments.

## *Experience with the Process*

We have used our modified process in several projects since the summer of 1996. We used Visio as a drawing tool and Lotus Notes as a repository for project deliverables as well as the data dictionary. The main benefit of using these tools was flexibility – in notation, in the kinds of deliverables in the repository and in the possibilities provided for modifying the process according to the size and character of the task.

After half a year, we had about 350 documents in the repository with the following distribution: task context, 36%; note, 15%; use case model, 10%; system interaction model, 14%; domain model, 4%; operation model, 5%; object interaction model, 3%; class model, 6%; class, 5%; and data types, 1%. The relatively large number of task context documents can be explained by the fact that some minor increments were sufficiently defined by their task context documents together with note documents and did not need to aggregate a full set of analysis and design deliverables.

The benefits of the object-oriented definition of the software development process are:

- It is robust and easy to use. Small increments typically result in a subset of deliverable classes: task context, plan, source code, user documentation and perhaps several design documents. The quality-assurance methods guarantee consistency between deliverables. With larger increments, the number of kinds of deliverables can be increased and always reflects project state and any specific requirements .
- The process provides good management support for incremental development. Each increment is defined in a single document that exists throughout the increment's life cycle.
- The process is flexible and applicable to tasks with a wide range of characteristics. By simply defining or redefining the methods and attributes of the deliverable classes, the process can be easily adapted for use with different kinds of projects.
- The process definition is applicable to different solution frameworks and organization cultures. For example, our organization is now implementing Microsoft Solutions Framework, which defines a team model, a process model and an application model. Only minor changes (mostly in terminology) are necessary in order to make the change from the object-oriented Fusion process to the MSF process model.

## *Discussion - Other Approaches*

Other possible approaches to the process definition are a workflow model and an object-oriented model in which activities are objects, tasks are object operations and deliverables are operation postconditions.

A workflow model separates activities and deliverables. In general, such a definition cannot cover all the possible combinations of activities and deliverables without becoming overly complex. It is an advantage that our definition focuses on the deliverables and that the choice of appropriate activity is left mostly up to the judgement of the developer and depends on the specific situation.

An object-oriented model with activities as objects is already used in several methodologies, however, we can see some potential problems:

- Quality issues. It is usually easier to define and ensure the quality of a deliverable rather than the quality of an activity.
- Usability issues. If you have to work with incomplete information during the planning phase, as is typically the case, it is usually easier to determine which deliverables have to be produced than to determine which activities will lead to the creation of the deliverables.
- Fail-safe issues. If an activity is omitted in the project plan, there is usually no warning during the planning phase that something is missing. In a deliverable-based approach, the quality-assurance methods guarantee consistency between deliverables.

## *Conclusions*

We have described an object-oriented model for the Fusion development process. The main artifacts of our model are deliverables, which are modeled as objects with constructors and quality-assurance methods and a number of specific attributes. We have described our experience with the model, which we find flexible, robust and easy to use in general.

## *References*

[1] Booch, G: Object Solutions, Addison-Wesley Publishing Company, 1996

[2] Coleman, D. et al.: Object-Oriented Development: the Fusion method, Prentice Hall 1994

[3] Hruby, P.: An Object Model for a Product Based Development Process, ECOOP'97 Workshop on Modeling Software Processes and Artifacts, 1997.

[4] Malan, R., Letsinger, R., Coleman, D.: Object-Oriented Development at Work: Fusion in the Real World, Prentice Hall 1996

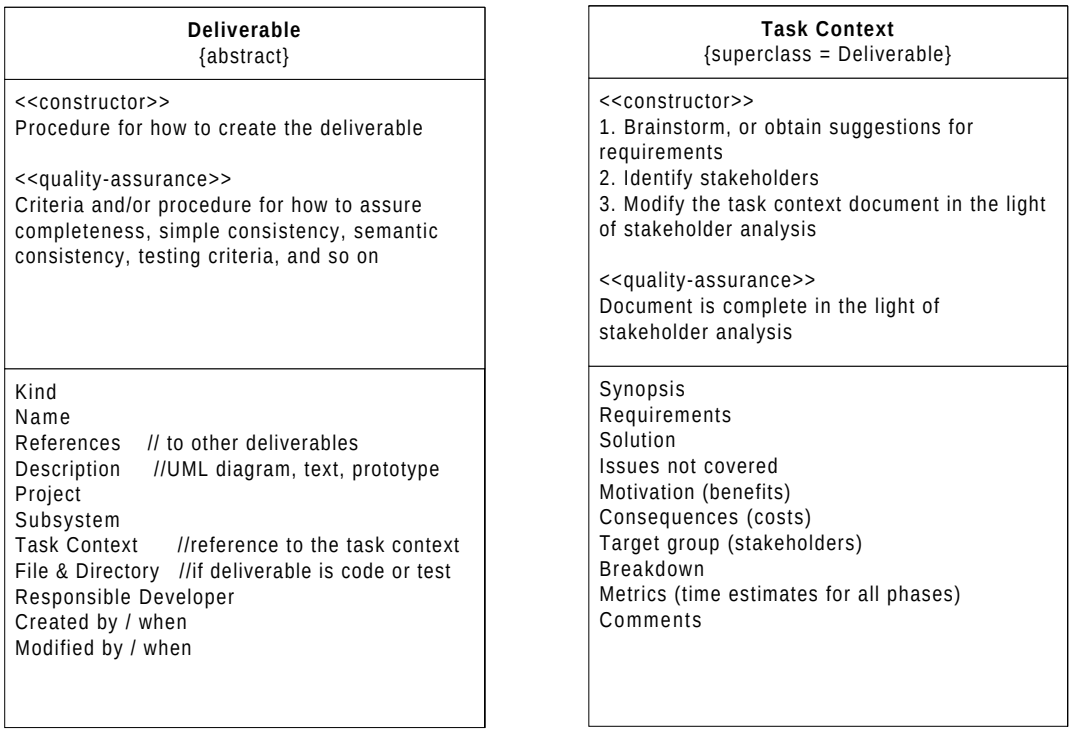


Fig. 1. Class specifications for the Deliverable and Task Context classes. The Task Context class is inherited from the Deliverable class; therefore it also contains all methods and attributes of the Deliverable class.

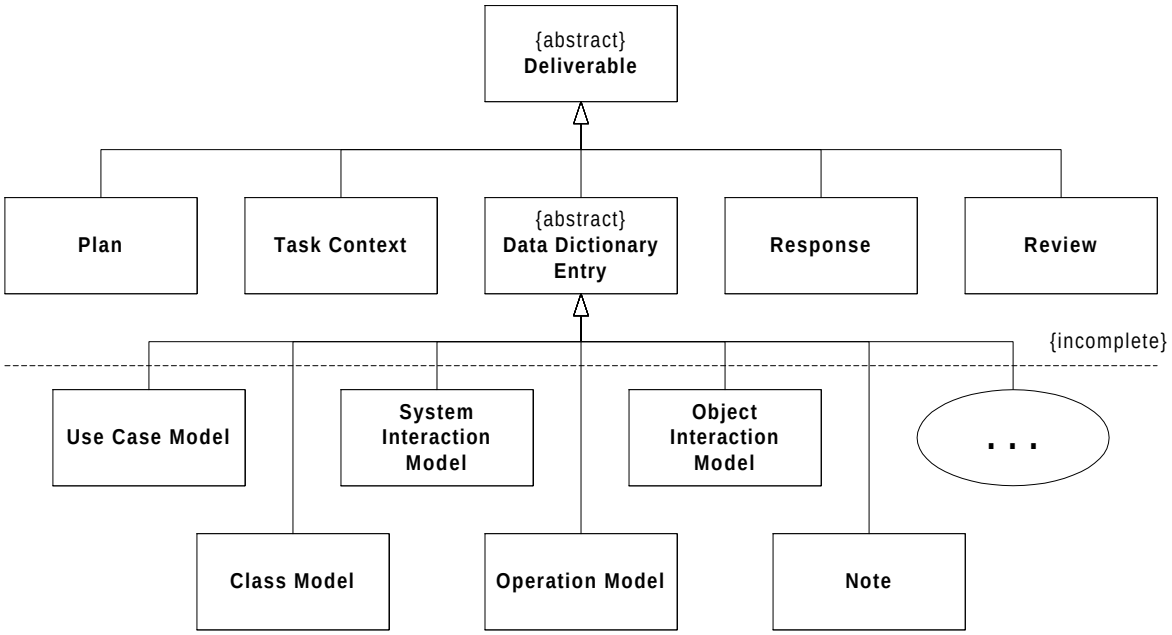


Fig. 2. The inheritance diagram illustrating classes of deliverables used in the development process. The fact that the inheritance tree is incomplete allows for flexibility throughout the process and for exact matching of project deliverables to different kinds of processes. The Data Dictionary Entry class has abstract methods defined in derived classes.

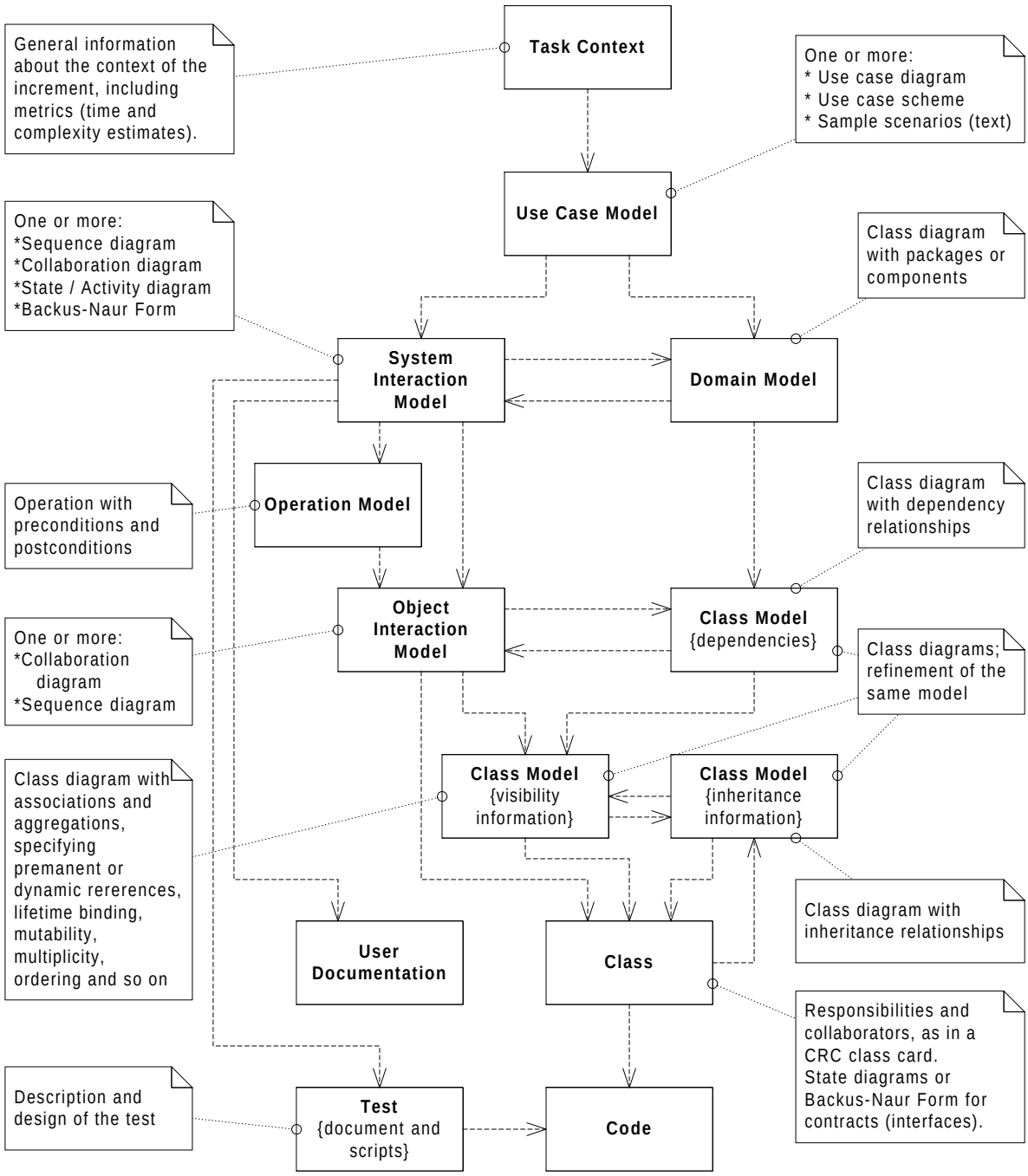


Fig. 3. Typical dependencies between deliverables (design process). Relationships do not necessarily determine the time sequence in which the documents are created. For example, deliverables of the Class class are established during the early phases of development but initially contain only responsibilities; the more complete description, which includes contracts, method signatures, and so on is determined during the design phase from the class model and the interaction model. The graph does not illustrate the design phases because the problem of the design phases is slightly more complicated. For example, if an increment is a new component inside of the product, then the operation model is part of the analysis because preconditions and postconditions are an important part of *what* the system does. If the increment has an interface to the user, then the operation model is part of the design because the user is typically satisfied with use cases or system interaction models and does not care about preconditions and postconditions. Furthermore, many deliverables are established in one phase but completed in another phase. Design phases are important, but they are better defined by what information the model contains, rather than by which deliverables are produced.

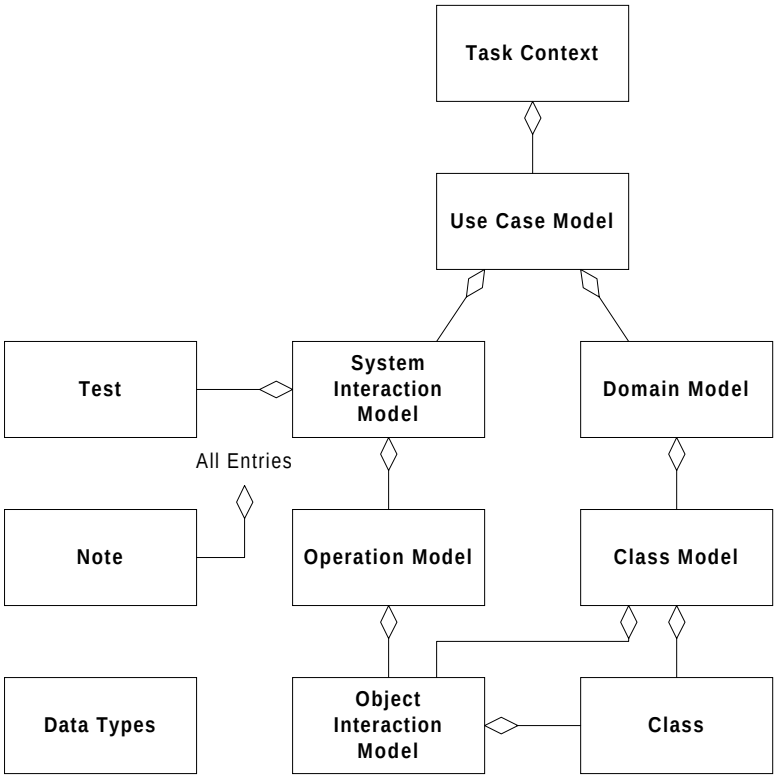


Fig. 4. References between analysis and design deliverables. Relationships are navigable in both directions. The figure illustrates only typical relationships; in principle, a deliverable can have bidirectional links to any other document or documents in order to enable maximal flexibility. For example, the object interaction model was often broken down into another object interaction model, or some tasks did not require the use of an operation model and links were directed to a system interaction model instead.