Source: <http://agilemodeling.com/essays/fdd.htm>

Top of Form

Feature-Driven Development (FDD) is a client-centric, architecture-centric, and pragmatic software process. The term "client" in FDD is used to represent what Agile Modeling (AM) refers to as [project stakeholders](http://agilemodeling.com/essays/activeStakeholderParticipation.htm#Stakeholders) or [eXtreme Programming (XP)](http://agilemodeling.com/essays/agileModelingXP.htm) calls customers. FDD was first introduced to the world in 1999 via the book [Java Modeling In Color with UML](http://www.amazon.com/exec/obidos/ASIN/013011510X/ambysoftinc/), a combination of the software process followed by Jeff DeLuca's company and Peter Coad's concept of features. FDD was first applied on a 15 month, 50-person project for a large Singapore bank in 1997, which was immediately followed by a second, 18-month long 250-person project. A more substantial description is published in the book [A Practical Guide to Feature-Driven Development](http://www.amazon.com/exec/obidos/ASIN/0130676152/ambysoftinc/) as well as the [Feature Driven Development](http://www.featuredrivendevelopment.com/) web site.

As the name implies, [features](http://agilemodeling.com/artifacts/feature.htm) are an important aspect of FDD. A feature is a small, client-valued function expressed in the form Top of Form

 A feature is a small, client-valued function expressed in the form Top of Form

 A feature is a small, client-valued function expressed in the form

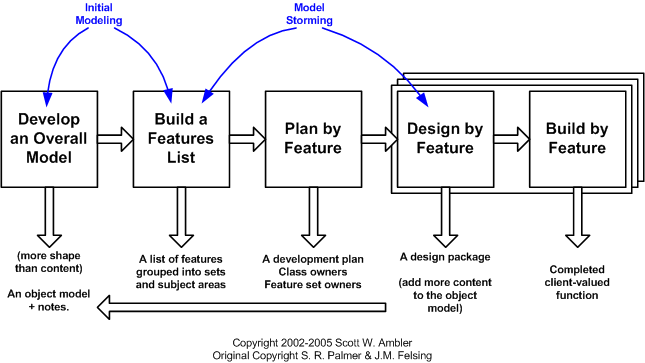
Bottom of Form

Bottom of Form

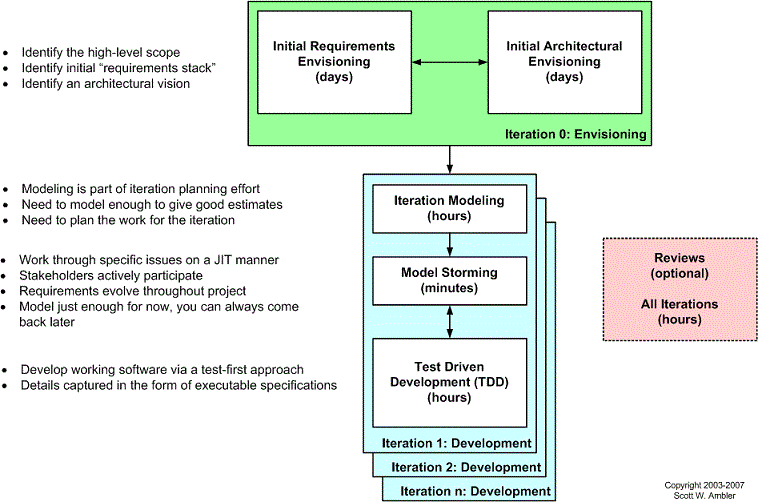
As you see in [Figure 1](http://agilemodeling.com/essays/fdd.htm#Figure1) there are five main activities in FDD that are performed iteratively. The first is Develop An Overall Model, the initial result being a high-level object model and notes. At the start of a project your goal is to identify and understand the fundamentals of the domain that your system is addressing, and throughout the project you will flesh this model out to reflect what you're building. The second step is Build A [Features List](http://agilemodeling.com/artifacts/feature.htm#Figure1), grouping them into [related sets and subject areas](http://agilemodeling.com/artifacts/feature.htm#Figure2FeatureSets). These first two steps map to the [initial envisioning effort of AMDD](http://agilemodeling.com/essays/amdd.htm#InitialModeling) (see [Figure 2](http://agilemodeling.com/essays/fdd.htm#Figure2)). Next you Plan By Feature, the end result being a development, the identification of class owners (more on this in a minute), and the identification of feature set owners. The majority of the effort on an FDD project, roughly 75%, is comprised of the fourth and fifth steps: Design By Feature and Build By Feature. These two activities are exactly

what you'd expect, they include tasks such as detailed modeling, programming, testing, and packaging of the system.Bottom of Form

**Figure 1. The FDD project lifecycle.**



**Figure 2. The lifecycle of AMDD.**



|  |  |
| --- | --- |
| An FDD project starts by performing the first three steps in the equivalent of the DAD's Inception phase or XP's "iteration 0", the goal being to identify the scope of the effort, the [initial architecture](http://agilemodeling.com/essays/initialArchitectureModeling.htm), and the initial high-level plan. Construction efforts occur in two-week (or less) iterations, similar to XP or DAD teams, with the team iteratively working through all five steps as needed. As with other agile software development processes, systems are delivered incrementally by FDD teams.  There are six primary roles on an FDD project: Project Manager, Chief Architect, Development Manager, Chief Programmer, Class Owner, and Domain Expert. An individual will take on one or more roles on a project as you would expect. The concept of a class owner is where FDD differs from XP. XP includes a practice called [Collective Ownership](http://agilemodeling.com/practices.htm#CollectiveOwnership) the idea of which is that any developer can update any artifact, including source code, as required. FDD takes a different approach in that it assigns classes to individual developers, so if a feature requires changes to several classes then the owners of those classes must work together as a feature team to implement it. Just like programming pairs will [model storm](http://agilemodeling.com/essays/modelStorming.htm) to think something through before they code it, so will feature teams. | [Practical Guide to FDD](http://www.amazon.com/exec/obidos/ASIN/0130676152/) |

FDD also defines a collection of supporting roles, including:

* Domain Manager
* Release Manager
* Language Guru
* Build Engineer
* Toolsmith
* System Administrator
* Tester
* Deployer
* Technical Writer

FDD's five steps are supported by several practices. The first is domain object modeling, the creation of a high-level class diagram and supporting artifacts that describes the problem domain. Developing by feature and individual class ownership are also good practices, as is having developers work together in feature teams. Inspections are an important aspect of FDD. FDD also insists on regular builds, similar to XP, and configuration management. Finally, FDD promotes a best practice called reporting/visibility of results, similar to XP and AM's philosophy of [open and honest communication](http://agilemodeling.com/principles.htm#OpenAndHonestCommunication).

How would Agile Modeling (AM) be applied on an FDD project? The principles and practices can be clearly applied to FDD's two modeling-oriented steps - develop an overall model and design by feature. The only apparent mismatch between the two processes is FDD's practice of class ownership and AM's practice of [collective ownership](http://agilemodeling.com/practices.htm#CollectiveOwnership), but I would argue that this isn't the case. FDD's practice pertains to coding but does not to modeling, on a FDD project people work together in teams to model, along the lines of AM's model with others practice, and therefore several people will be working on your shared collection of modeling artifacts.