Agile Modeling

Feature Driven Development (FDD) and Agile Modeling

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 or eXtreme Programming (XP) calls customers. FDD was first introduced to the world in 1999 via the book Java Modeling In Color with UML, 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 as well as the Feature Driven Development web site.

As the name implies, features are an important aspect of FDD. A feature is a small, client-valued function expressed in the form <action><result><object>. For example, "Calculate the total of a sale", "Validate the password of a user", and "Authorize the sales transaction of a customer". Features are to FDD as use cases are to the Rational Unified Process (RUP) and user stories are to Scrum - they're a primary source of requirements and the primary input into your planning efforts.

As you see in Figure 1 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, grouping them into related sets and subject areas. These first two steps map to the initial envisioning effort of AMDD (see Figure 2). 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.

Figure 1. The FDD project lifecycle.

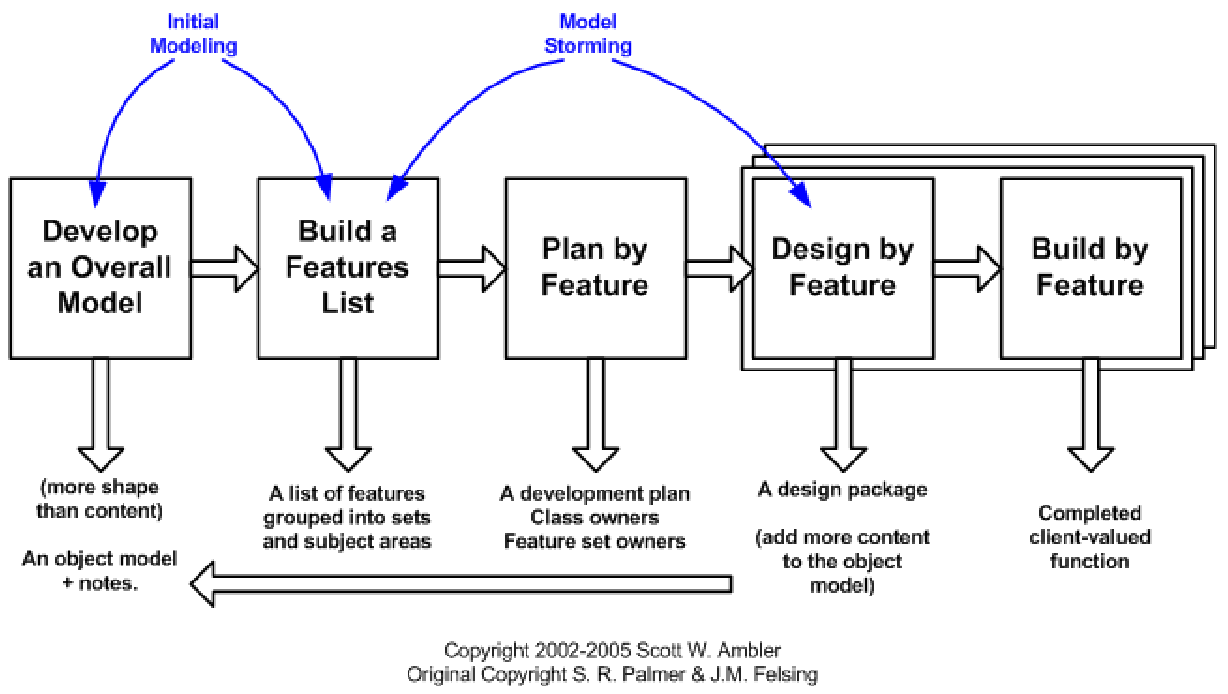
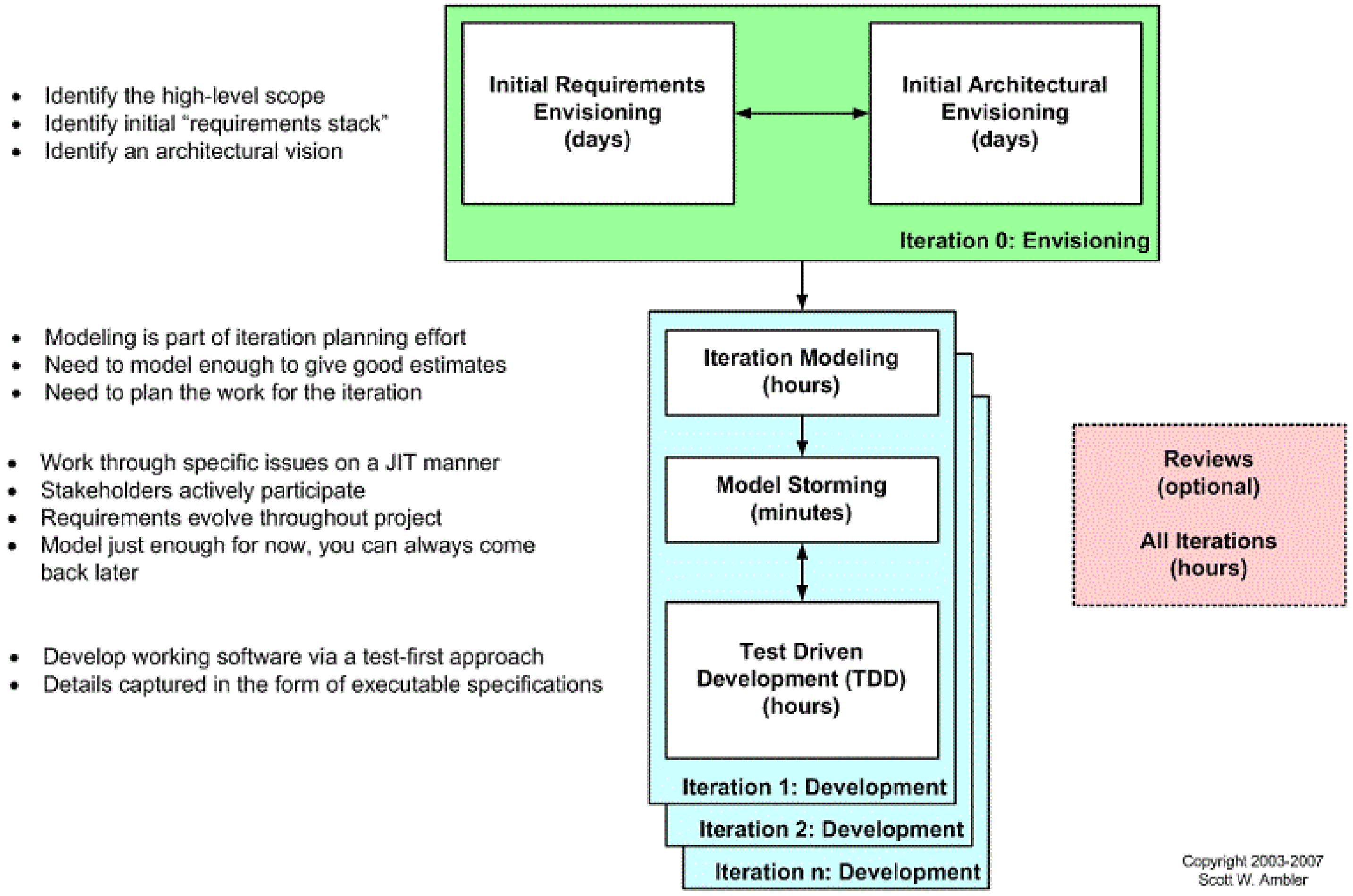
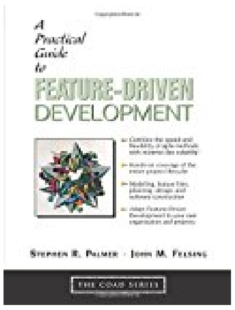

Initial Modeling
Model Storming
Develop an Overall Model
Build a Features List
Plan by Feature
Design by Feature
Build by Feature
(more shape than content)
An object model + notes.
A list of features grouped into sets and subject areas
A development plan
Class owners
Feature set owners
A design package
(add more content to the object model)
Completed client-valued function
Copyright 2002-2005 Scott W. Ambler
Original Copyright S. R. Palmer & J.M. Felsing

Figure 2. The lifecycle of AMDD.



Identify the high-level scope
Identify initial "requirements stack"
Identify an architectural vision
Initial Requirements Envisioning (days)
Initial Architectural Envisioning (days)
Iteration 0: Envisioning
Modeling is part of iteration planning effort
Need to model enough to give good estimates
Need to plan the work for the iteration
Iteration Modeling (hours)
Work through specific issues on a JIT manner
Stakeholders actively participate
Requirements evolve throughout project
Model just enough for now, you can always come back later
Model Storming (minutes)
Develop working software via a test-first approach
Details captured in the form of executable specifications
Test Driven Development (TDD) (hours)
Iteration 1: Development
Iteration 2: Development
Iteration n: Development
Reviews (optional)
All Iterations (hours)
Copyright 2003-2007 Scott W. Ambler

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, 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 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 to think something through before they code it, so will feature teams.

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.

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, 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.

Recommended Reading

This book, Choose Your WoW! A Disciplined Agile Delivery Handbook for Optimizing Your Way of Working, is an indispensable guide for agile coaches and practitioners to identify what techniques - including practices, strategies, and lifecycles - are effective in certain situations and not as effective in others. This advice is based on proven experience from hundreds of organizations facing similar situations to yours. Every team is unique and faces a unique situation, therefore they must choose and evolve a way of working (WoW) that is effective for them. Choose Your WoW! describes how to do this effectively, whether they are just starting with agile/lean or if they're already following Scrum, Kanban, SAFe, LeSS, Nexus, or other methods.

The Object Primer 3rd Edition: Agile Model Driven Development with UML 2 is an important reference book for agile modelers, describing how to develop 35 types of agile models including all 13 UML 2 diagrams. Furthermore, this book describes the fundamental programming and testing techniques for successful agile solution delivery. The book also shows how to move from your agile models to source code, how to succeed at implementation techniques such as refactoring and test-driven development(TDD). The Object Primer also includes a chapter overviewing the critical database development techniques (database refactoring, object/relational mapping, legacy analysis, and database access coding) from my award-winning Agile Database Techniquesbook.

DISCIPLINED AGILE | Agile Modeling | Agile Data | Enterprise Unified Process | @scottwambler