SSW-555: Agile Methods for Software Development

Feature Driven Design (FDD)

Instructor: Prof Zhongyuan Yu

School of Systems and Enterprises

Stevens Institute of Technology

This Lecture is based on material from Prof Jim Rowland

Today's topic

- Motivation: Large Projects
- Origins of FDD
- FDD Roles
- FDD Processes
- FDD Practices
- Similarities and differences from other Agile Methods

Start with some interesting quotations …

- "Kanban is the science of not trying to do too much at once" -- Stephen Palmer, 2012
- "We try to solve the problem by rushing through the design process so that enough time is left at the end of the project to uncover the errors that were made because we rushed through the design process" -- Glenford Myers (via Jeff De Luca)
- "Agile software development is about iteration not oscillation" -- Jim Highsmith (paraphrased), Agile 2009

Motivation: Some Cocomo estimates

- So far, we've looked at applying Agile Methods to relatively small projects.
- How do we scale Agile Methods to larger projects?
  - Many more developers working much longer times.
  - Techniques we've discussed so far won't work for very large projects.

KLOC	Effort (Staff Months)	Calendar Months	Software Construction Staff Months	Software Construction Calendar Months	# Staff
1	0.5	2	0.4	1.2	0.3
10	5.8	4.4	4.4	2.7	1.6
100	64.2	9.7	48.8	6	8.1
1,000	708.4	21.3	538.4	13.3	40.4
10,000	7815.5	47.1	5939.7	29.4	201.7

Agile Manifesto (2001)

- We are uncovering better ways of developing software by doing it and helping others do it

- Through this work we have come to value:
  - **Individuals and interactions** over processes and tools.
  - **Working software** over comprehensive documentation.
  - **Customer collaboration** over contract negotiation.
  - **Responding to change** over following a plan.

- That is, while there is value in the items on the right, we value the items on the left more.

FDD's view on the Agile Manifesto

- **Responding to change** over following a plan
  - **doesn't** mean you **shouldn't** plan
  - **instead**, planning is important
  - **but** you must be willing to **change the plan**

- **Working software** over comprehensive documentation
  - **doesn't** mean you **shouldn't** write documentation
  - **instead**, documentation is important and required
  - **but** documentation without working code is a path to failure

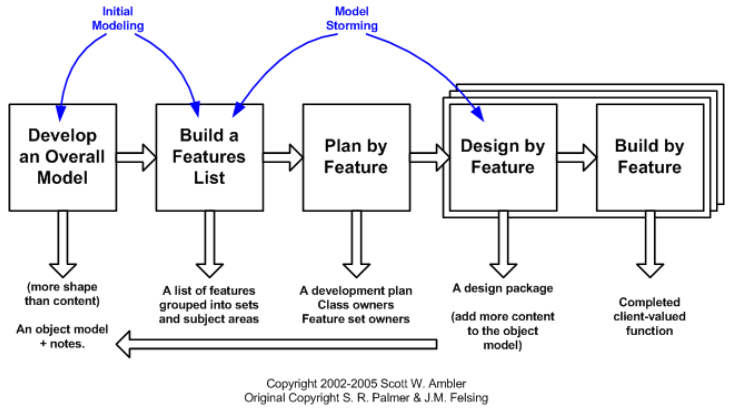
FDD’s view on Process

- A system for building systems is necessary in order to scale to larger projects
- A simple, but well-defined process works best
- Process steps should be logical and their worth immediately obvious to each team member
- “Process pride” can keep the real work from happening
- Good processes move to the background so team members can focus on results
- Short, iterative, feature-driven life cycles are best

Source: pp. 273, Jim Highsmith, Agile Software Development Ecosystems

FDD Origins

- Developed in 1997 by Jeff DeLuca on *Singapore project* with help from Peter Coad and Stephen Palmer
- FDD depends on ***strong OO modeling*** at ***start of project***
  - Assumes overall model of the system at the beginning
- Adopted short iterations to address shorter business cycles
- Lightweight process specification relative to plan based methods used in the past, but heavier weight than some Agile methods



Singapore Project

- Banking domain – lending instruments
- Failed 2-year project by a "well-known systems integration firm"
  - 3500 pages of "useless cases" (vs. use cases)
  - Hundreds of OO classes
  - Thousand of attributes
  - No code

- Using FDD, they successfully developed 2000 features in 15 months with 50 staff
  - Developed initial model in 1 month
  - Planned features in 2 weeks

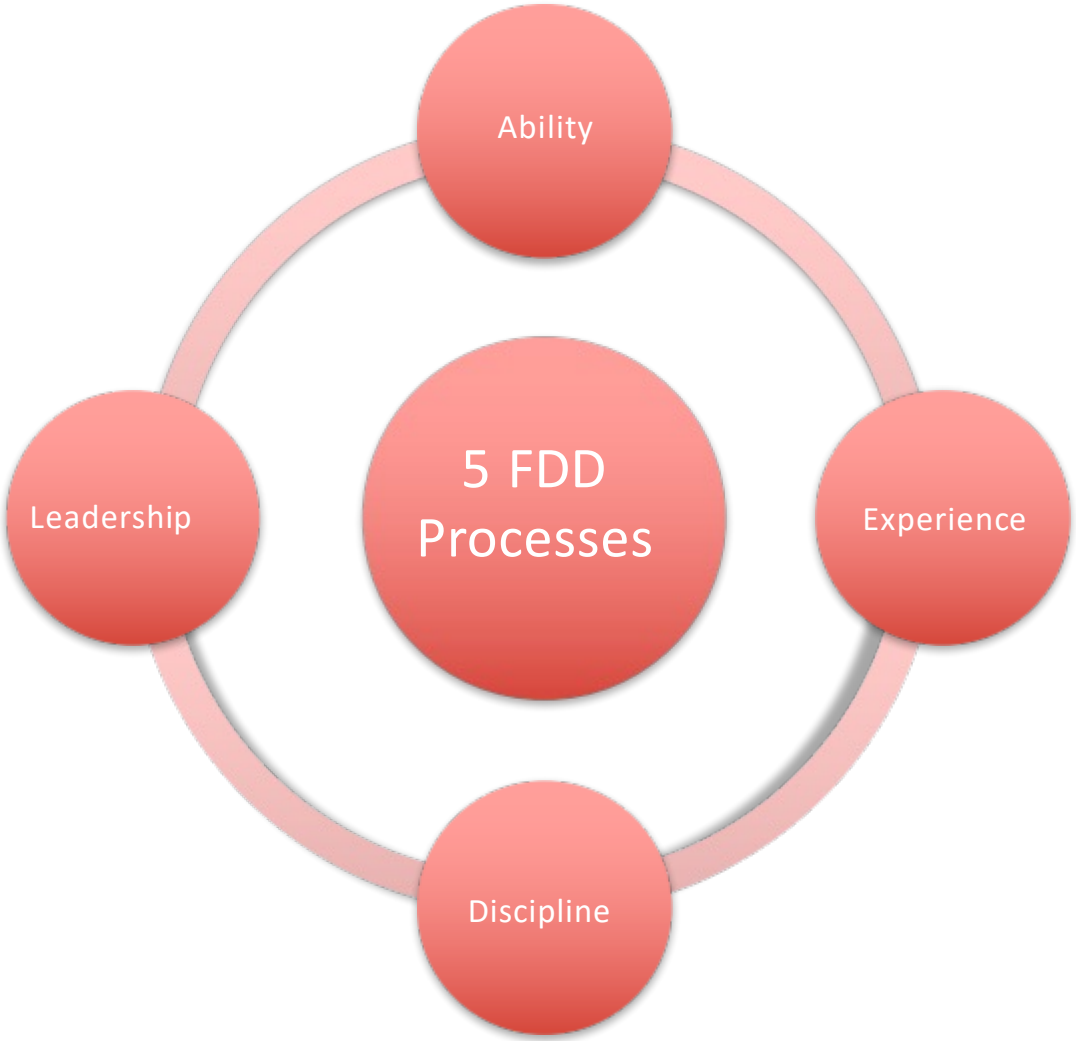
FDD Features

- Cross-functional team
- Focus on collaboration
  - Developers working closely with customers
- Time boxed activity
- Start project with:
  - Shared understanding of the project
  - Shared vocabulary between developers and customers
  - Conceptual framework
  - Overall Model of the system

Roles in FDD

- **Domain Experts** – voice of customer
- **Project Manager** (DeLuca) – administrative lead
- **Chief Architect** (Coad) – responsible for overall design
- **Development Manager** (Palmer) – team management
- **Chief Programmers** – lead teams in design of features
- **Feature Teams** – temporary groups of developers implementing features
- **Class Owners** – **developers who own individual classes**
  - Different from XP's collective code ownership

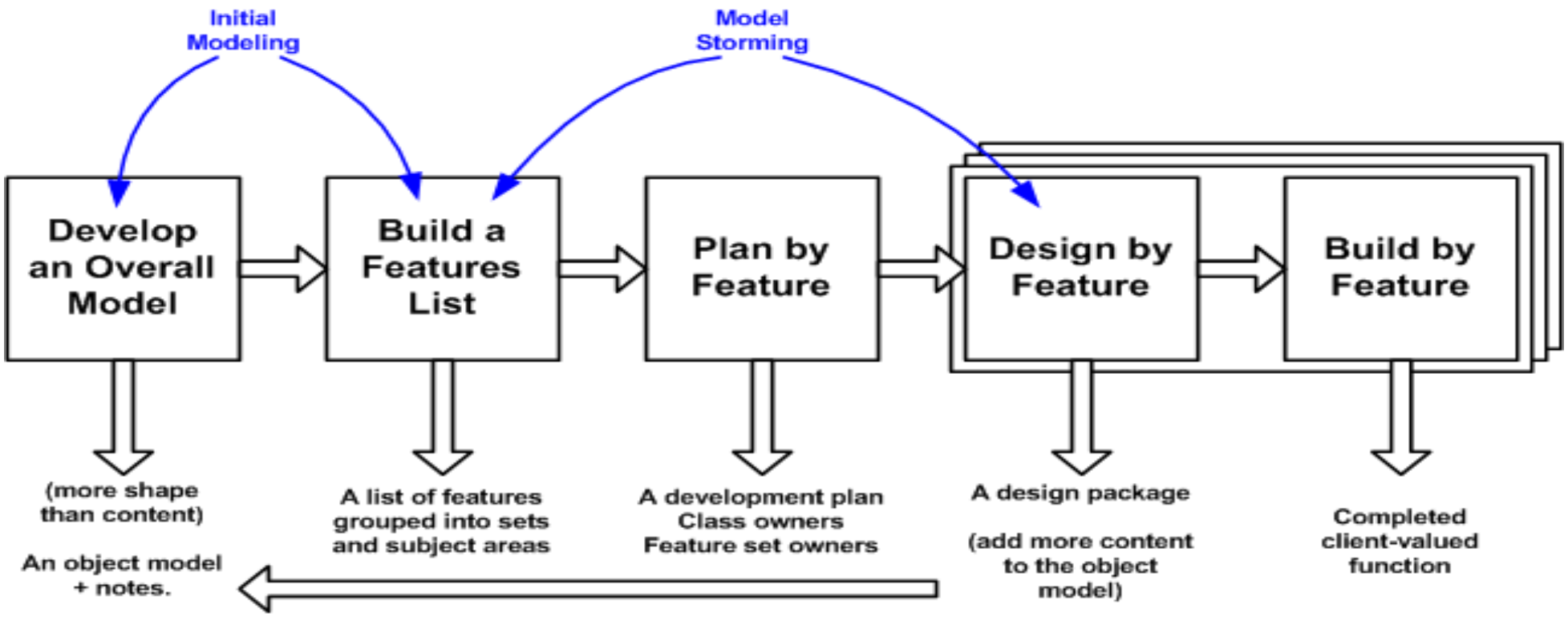
Teams also require …



The 5 FDD Processes

JEDI (Just Enough Design Initially) approach

Not BDUF (Big Design Up Front) as in Plan driven approaches



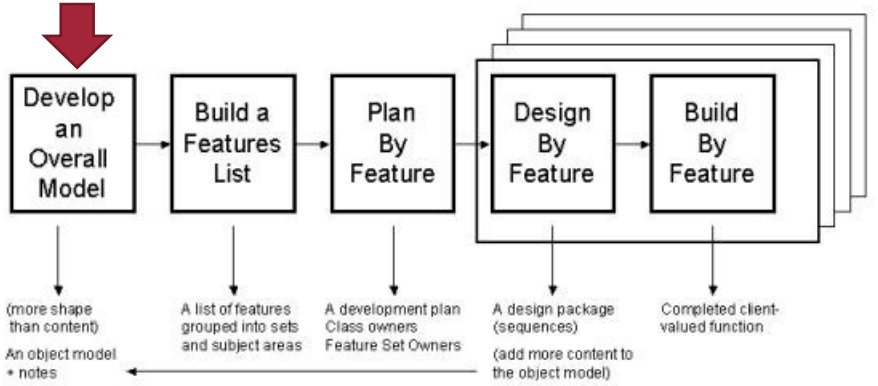
http://agilemodeling.com/essays/fdd.htm

General effort guidelines

Process	Initial	Ongoing
1 Develop an Overall Model	10%	4%
2 Build a Features List	4%	1%
3 Plan by Feature	2%	2%
4 Design by Feature	77%	
5 Build by Feature		

Process 1: Develop an Overall Model

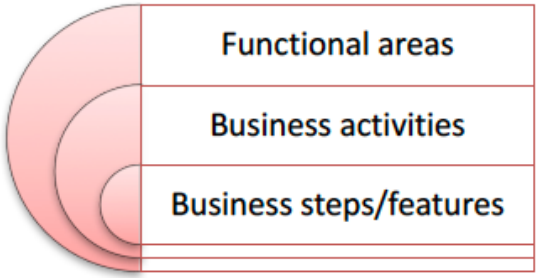
1. Form the modeling team (led by Chief Architect)
   - Includes Domain experts, Chief Programmers
2. Perform domain walk-through
3. Study relevant documents to understand domain knowledge
4. Develop the model (led by Chief Architect)
   - Define overall “**shape**" of the model, rather than the details
   - More details will be added during the Design by Feature process
5. Refine the overall object model
6. Write model notes
7. Internal and external assessment

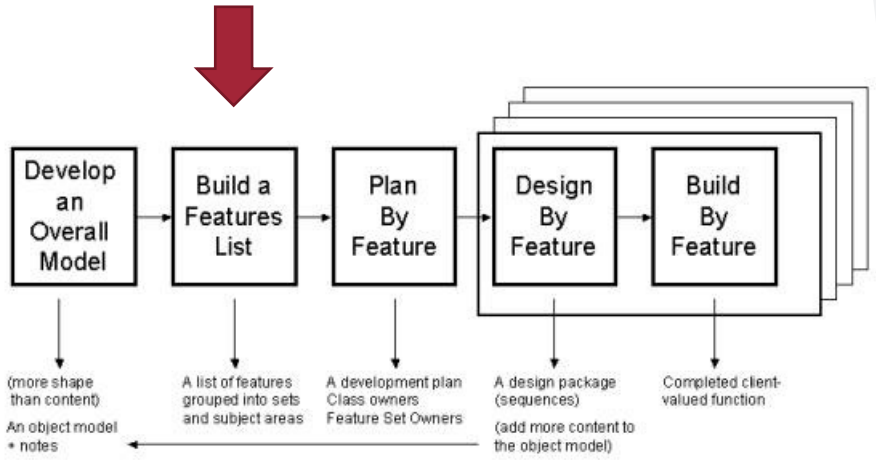


Process 2: Build a Features List

1. Form the features list team (Chief Programmers) Goal is to define all the features (product backlog)

2. Build features list – oriented to the business needs Hierarchical, not flat, like user stories





Features should be small enough to implement in 10 days

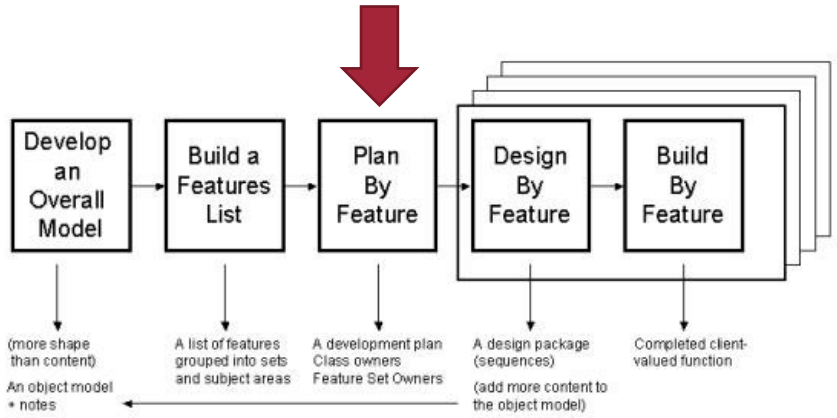
Describe each feature with the template:

<action> <result> <object>

E.g., calculate the total of a sale

Process 3: Plan by Feature

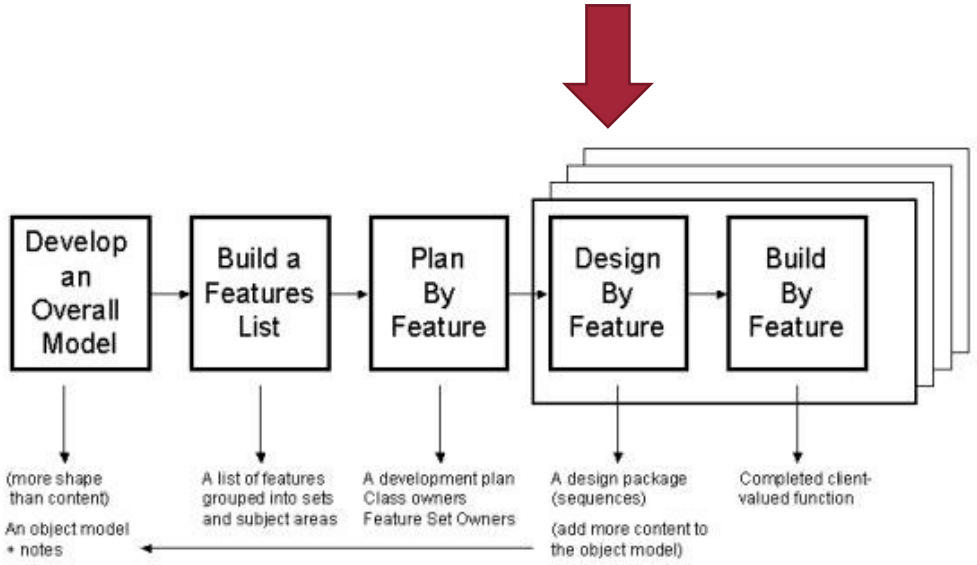
Goal is to create a plan to deliver all features



1. Form the planning team
   - (Development Manager, Chief Programmers)
2. Determine the development sequence
   - Some dependencies are obvious, but others are more subtle
   - May depend upon resource availability
3. Assign business activities to Chief Programmers
4. Assign classes to developers
   - Unlike XP's collective ownership, classes are owned by individual developers

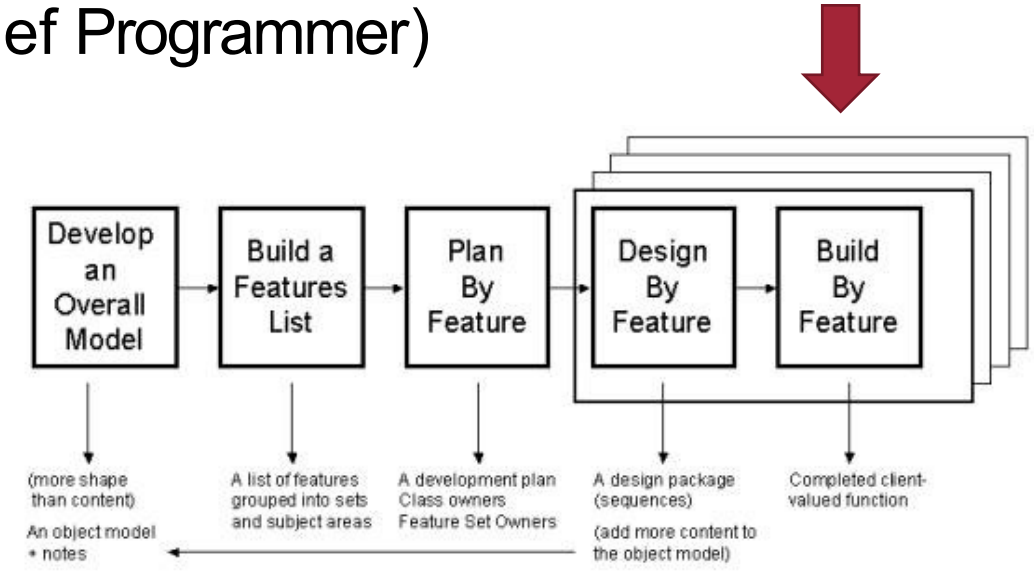
Process 4: Design by Feature

1. Form feature team (Chief Programmer, developers)
   - Dynamic team, formed for this feature
2. Domain walk-through (domain expert)
3. Study the referenced documents (team)
4. Develop the UML sequence diagrams (team)
   - Done as group activit
   - Scribe records notes
5. Refine the object model (Chief Programmer)
6. Write class and method prologues (developers)
7. Design inspection (team)



Process 5: Build by Feature

1. Implement classes and methods (class owner)
2. Code inspection (feature team)
3. Unit test (class owner)
4. Promote to the build (class owner with Chief Programmer)
5. Disband the team

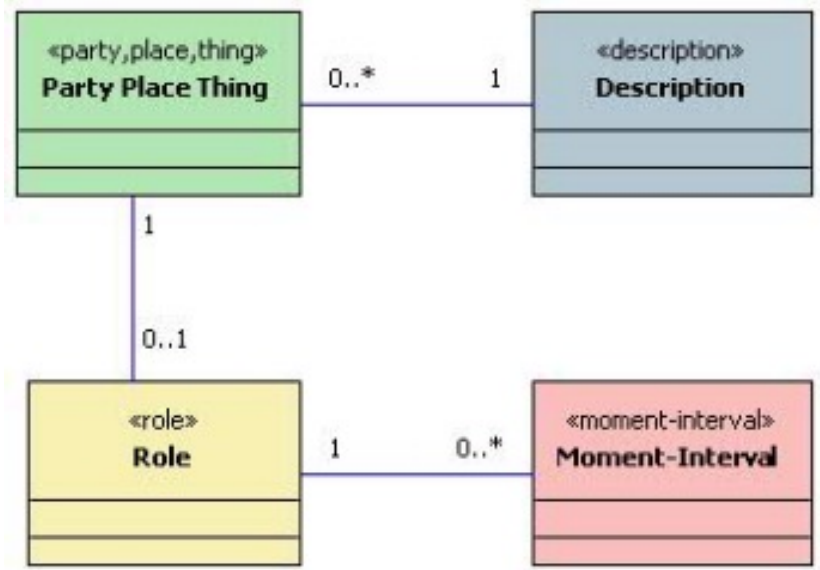


FDD Practices

- Domain Object Modeling
- Developing by Feature
- Individual Class/Code Ownership
- Feature Teams
- Inspections
- Regular Builds
- Configuration Management
- Reporting / Visibility of Results

Domain object modeling

- Construct class diagrams for ***all*** significant object types
  - Define the class names and description, but not all details
- Supplement with UML sequence diagrams showing interactions between objects
- Coad developed modeling in color during Singapore project to standardize class patterns



Developing by feature

A feature is a small, client valued function expressed in the form:
<action> <result> <object>

Examples:

Calculate the total of a sale

Assess the performance of a salesman

Validate the password of a user

Feature template provides clues to operations and classes to which they should be applied

'Calculate the total of a sale' suggests:

calculateTotal() operation in a Sale class

'Assess the performance of a salesman' suggests:

assessPerformance() operation in a Salesman class

Individual class/code ownership

- Each class is owned by one developer
  - Opposite of collective ownership in XP
- Advantages:
  - Owner maintains conceptual integrity of class
  - Each class has an expert associated with it
  - Owner can implement changes faster than anyone else
  - Owners can take pride in their classes
- Disadvantages
  - Dependencies between classes (owner A needs owner B to make changes to their classes)
  - Risk of loss of owners during development

Feature Teams

- Each feature is owned by a Feature Team
  - Team is created dynamically to design and build a feature
  - Disbanded after the feature is complete
- Each team consists of 2-5 class owners (developers) led by a Chief Programmer
- Each team contains all the owners it needs to create the feature
  - Everyone needed to develop the feature is on the team
  - Don’t depend on anyone outside the team
- Owners may be on multiple feature teams at the same time
- Chief programmers may also be class owners

Inspections

- FDD inspections usually performed by feature teams on their own features
  - FDD suggests inspections over Pair Programming
- Designs and code are inspected by team members
- Advantages:
  - Most effective technique for discovering defects
  - Provide educational benefits: domain knowledge, technical skills Promotes and enforces standards

Regular builds

- Some teams build weekly, others daily and others continuously
- Advantages of regular builds:
  - Early detection of integration problems
  - Always have something to demonstrate to client, even if not doing frequent releases
- Regular build process can be enhanced to:
  - Generate documentation
  - Run audit and metric scripts to highlight potential problem areas and to check for standards compliance
  - Run automated regression tests
  - Construct new build and release notes listing new features added, defects fixed

Configuration management

- Keep code under configuration management to manage conflicts and maintain consistency

- Other items kept under configuration management:
  - Requirements documents
  - Analysis and design artifacts
  - Test cases, test harnesses and scripts and even test results
  - Version of the ***process*** you are using, any changes and adjustments that may be made during the construction and maintenance of the system
    - Useful for audits and measuring process effectiveness

Reporting/visibility of results

- Track by feature
- Roll up summaries to feature set level
  - Parking lot chart for executives
- Track progress with burn-up graph
  - Similar to burn-down graph

Track by feature

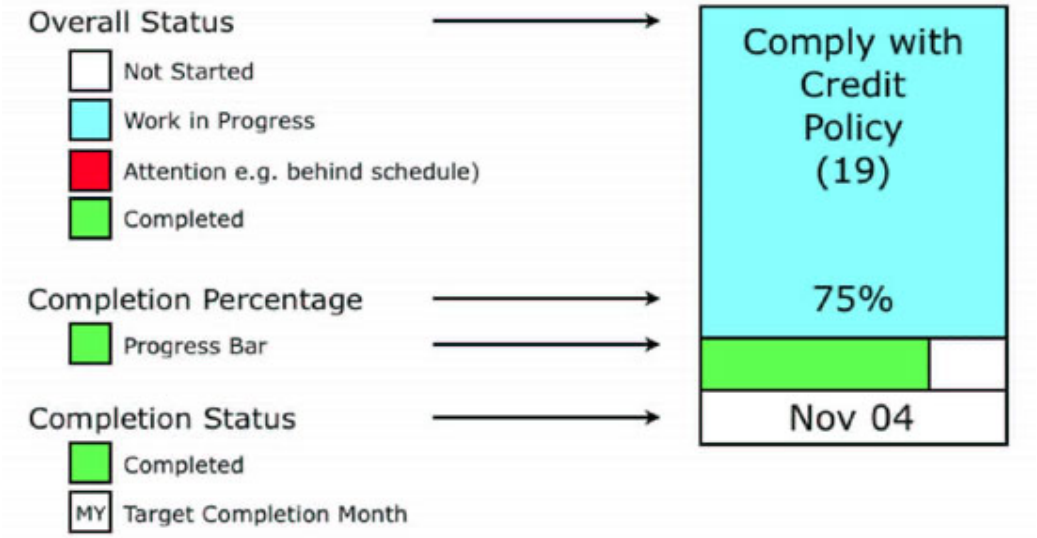
Explode Design Model (19)

ID	Description	Walkthrough		Design		Design Inspection		Development		Code Inspection		Promote to Build	
		Plan	Actual	Plan	Actual	Plan	Actual	Plan	Actual	Plan	Actual	Plan	Actual
DP036	add an Exploded Item to an Exploded Item List	03/10/2000	03/10/2000	05/10/2000	06/10/2000	06/10/2000	09/10/2000	10/10/2000	10/10/2000	12/10/2000	12/10/2000	12/10/2000	12/10/2000
DP037	add a List of Exploded Item to an Exploded Item List	03/10/2000	03/10/2000	05/10/2000	06/10/2000	06/10/2000	09/10/2000	10/10/2000	10/10/2000	12/10/2000	12/10/2000	12/10/2000	12/10/2000
DP038	iterate through an Exploded Item List	03/10/2000	03/10/2000	05/10/2000	06/10/2000	06/10/2000	09/10/2000	10/10/2000	10/10/2000	12/10/2000	12/10/2000	12/10/2000	12/10/2000
DP039	explode the Design Items for a Design Product	03/10/2000	03/10/2000	05/10/2000	06/10/2000	06/10/2000	09/10/2000	10/10/2000	10/10/2000	12/10/2000	12/10/2000	12/10/2000	12/10/2000
DP040	determine the list of valid Design Models during an interval for a Design Product	03/10/2000	03/10/2000	05/10/2000	06/10/2000	06/10/2000	09/10/2000	10/10/2000		11/10/2000		12/10/2000	
		REMARKS: Blocked on domain expert response REMARKS: Still blocked on [illegible]											
DP041	determine if a Design Model contains any complex Design Items	03/10/2000	03/10/2000	05/10/2000	06/10/2000	06/10/2000	09/10/2000	10/10/2000	10/08/2001	12/10/2000	12/10/2000	12/10/2000	12/10/2000
DP042	determine if a Design Item is complex	03/10/2000	03/10/2000	05/10/2000	06/10/2000	06/10/2000	09/10/2000	10/10/2000	10/10/2000	12/10/2000	12/10/2000	12/10/2000	12/10/2000
DP043	explode the Design Items for a simple Design Model	03/10/2000	03/10/2000	05/10/2000	06/10/2000	06/10/2000	09/10/2000	10/10/2000	10/10/2000	12/10/2000	12/10/2000	12/10/2000	12/10/2000
DP044	explode the Design Items for a simple reversible Design Model												
DP045	explode the Design Items for a complex Design Model with one complex Design Item												
DP046	list the Design Models for a complex Design Item												
DP047	explode the Design Items for a complex Design Model with two complex Design Items												
DP048	select each combination of a Pair of Design Model Lists												
DP049	generate an Exploded Item List for a combination selection												
DP050	record the Exploded Item List for a Design Model												
DP051	recall the Exploded Item List for a Design Model												
DP052	explode the Design Items for a complex Design Model with multiple complex Design Items												
DP053	iterate through each combination of multiple Design Model Lists												
DP054	select each combination in user specified order of multiple Design Model Lists												

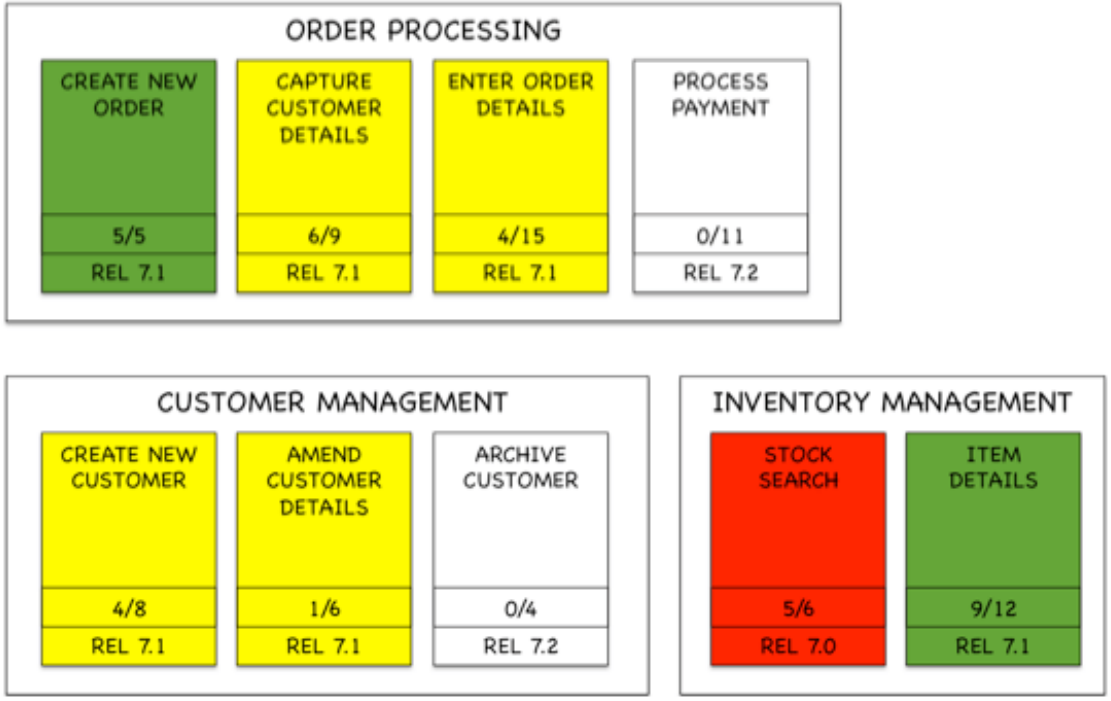
Progress sum for these features in business activity 'Explode Design Model': **39%**
Expected completion date: **Feb 2001**

https://www.methodsandtools.com/archive/archive.php?id=19p3

Feature set level --- Parking Lot Chart

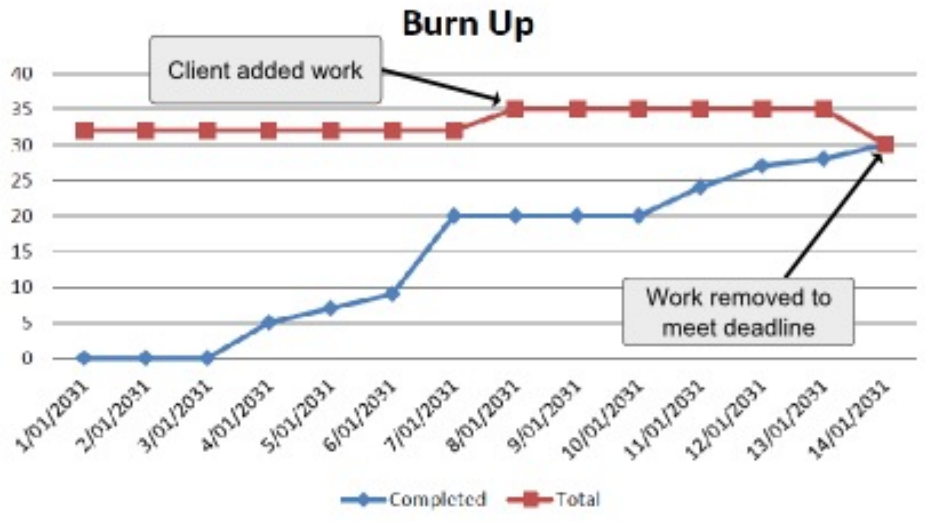


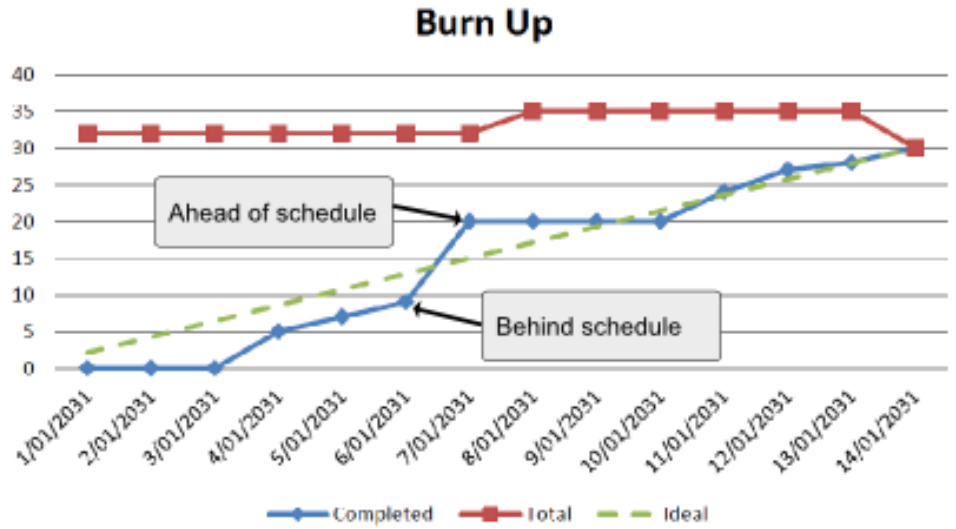
https://www.methodsandtools.com/archive/archive.php?id=19p3



https://insideproduct.co/parking-lot-diagram/

Burn up graph





https://www.clariostechnology.com/productivity/blog/whatisaburnupchart/

Similarities to other Agile Methods

- Short, time-boxed iterations
- Lightweight process
- Requirements described as features
- Customer participation in planning
  - but not as often, and at a higher level

Differences from other Agile Methods

- Up-front modeling – "Just Enough Design Initially (JEDI)", to avoid refactoring
  - Bias toward getting it right the first time
- FDD values documentation (models, notes, reference documents)
- More emphasis on progress tracking and reporting
- Dynamic feature teams
- Chief Programmers vs developers
- Design and code owned by individuals
- Code inspections/peer reviews instead of pair programming

Differences from other Agile Methods: Scheduling features

- XP/Scrum:
  - Customer adjusts priorities at beginning of each sprint/iteration
- FDD:
  - Customer's overall priorities are set early
  - *Business activities* are scheduled by the customer
    - Scheduled at month/year granularity
  - *Features* are scheduled by the technical team

STEVENS
INSTITUTE OF TECHNOLOGY
1870

THANK **YOU**

Stevens Institute of Technology
1 Castle Point Terrace, Hoboken, NJ 07030