chapter 4

Featu re- Drive n Development-Processes

Answers: How do we manage?

With good habits in using simple. well-defined processes. the process itself moves from foreground to background. Team members focus on results rather than process micro-steps. Progress accelerates. The team reaches a new stride. The team performs!

Coad. LeFebvre. De Luca [Coad 991

The Scope of Feature-Driven Develop-ment (FDD)

Mac: *If we use* FDD, will Lisa. *our project manager. have all the tools she needs to manage and control the entire software development project and all of its activities? I guess I'm asking what* FOO *covers and what it leaves as an exercise for the reader.*

Steve: FOO is *specifically targeted at the actual design and construction of the software system.* It *focuses only on the specific activities and outputs needed to design and write software .*

Mac: *There are a lot of other activities going on in a typical software development project* (Figure 4-1). *Why aren't they included, too? Are they not considered important?*

Steve: *Most organizations have workable processes in place for many of those activities. The exception* is *the actual construction of the software.* It *seems that once past the creation of initial., high-level requirements, the processes simply do not work well until we reach system testing. The actual design and building of the software* is *the big problem.* FDO is *aimed precisely at that core, problem area.*

That doesn't mean that the other activities aren't important. . They are extremely important, and we can certainly talk about them later However, the majority of the pain in a development project is felt in the core activities between initial requirements and system test.. If we cannot create code that does the right thing in the right way in a timely manner, then all the other activities are in vain, anyway.

Mac: *Some of the activities outside the scope of the* FDD *processes could produce input that* is *either used by* FOD *or that directly interacts with/impacts some of the* FDD *processes (eg, . initial requirements discussions, change request management,* or *the system, load and user acceptance testing) .* Is *that interaction with* FDO *expected, and how* is it *managed? .*

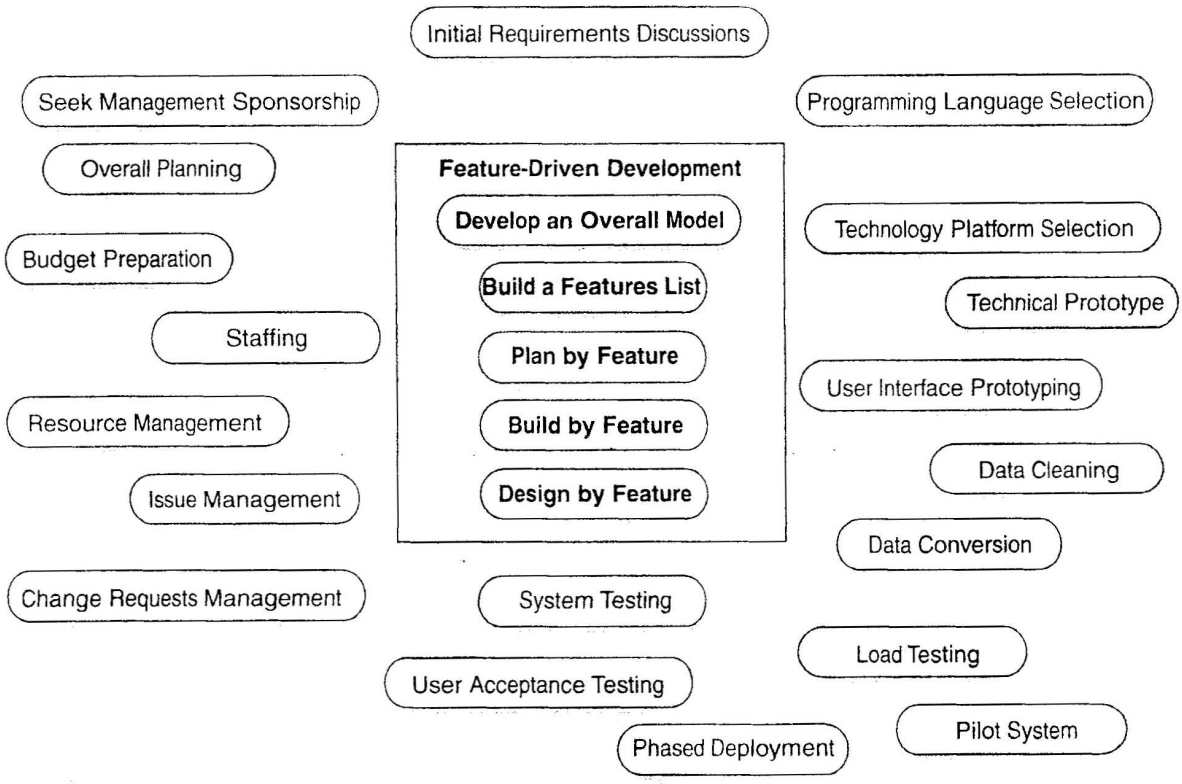

Figure 4-1
Some activities of a software development project

Steve: As we go through the FDD *process in more detail., you should be able to see how* FDD *works very neatly with the other important activities. You may even spot some opportunities to make some FDD-influencedimprovements to your specialty areas of system and user acceptance testing, and to requirements change contra!'.*

Mac: Okay. Let's look at the core FDD *process, then ...*

FDD in Four Sentences

FDD starts with the creation of a domain object model in collaboration with Domain Experts. Using information from the modeling activity and from any other requirements activities that have taken place, the developers go on to create a features list. Then a rough plan is drawn up and responsibilities are assigned. Now we are ready to take small groups of features through a design and build iteration that lasts no longer than two weeks for each group and is often much shorter-sometimes only a matter of hours-repeating this process until there are no more features.

FDD in a Nutshell

FDD consists of five processes (Figure 4-2). For the first process, domain and development team members work together under the guiding hand of an experienced object modeler (Chief Architect). Domain members present an initial high-level walkthrough of the scope of the system and its context. Then the domain members present more detailed walkthroughs of each area of the problem domain. After each walkthrough, the domain and development members work in small groups to produce object models for that area of the domain. Each small group presents its results to the team, and the team compares and discusses the various models, finally deciding on the most appropriate model for that domain area. The overall model shape is adjusted on the way, as necessary.

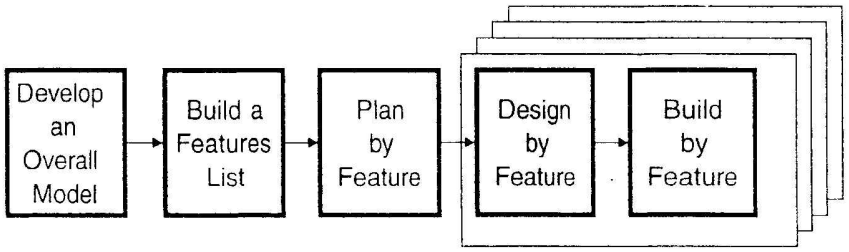

Figure 4-2
The five processes of FDD

Building on the knowledge gathered during the initial modeling activity, the team next constructs as comprehensive a list of features as it can (Figure 4-3). A *feature* is defined as a small, client-valued function expressed in the form: <action> <result> <object> (e.g., *"calculate the total* of a *sale").* Existing requirements documents, such as use cases or functional specs, are also used as input. The domain is reviewed, using a similar breakdown to that used by the Domain Experts when walking through the domain during the modeling activity. Within these domain areas, also called *major feature sets,* the features are grouped into feature sets. A feature set usually reflects a particular business activity. On initial compilation, the users and sponsors of the system review the feature list for validity and completeness.

The third process is to sequence the feature sets or major feature sets (depending on the size of the system) into a high-level plan and assign them to Chief Programmers. Feature sets are sequenced, depending on priority and dependencies. Also, the classes identified in the modeling activity are assigned to individual developers; the owner of a class is responsible for its development In large projects, major milestone timeframes may be set that time-box the completion of a number of feature sets. For example, a time box of three months might be set to complete the first few feature sets and formally demonstrate the results to upper management The team can also use the time box milestone to reset the plan formally and gain management approval for the change. For shorter projects, it's usually unnecessary to include this level of formality in the plan.

Processes 4 and 5 are the development engine room. A Chief Programmer selects a small group of features to develop over the next few days (no longer than two weeks). He or she identifies the classes likely to be *involved,* and the corresponding class owners form the feature team for this iteration. This feature team works out detailed sequence diagrams for the features and writes class and method prologues. The team then conducts a design inspection. After a successful inspection, the class owners add the actual code for their classes, unit test, integrate, and hold a code inspection. Once the Chief Programmer is satisfied, the completed features are promoted to the main build, and processes 4 and 5 repeat with the next group of features.

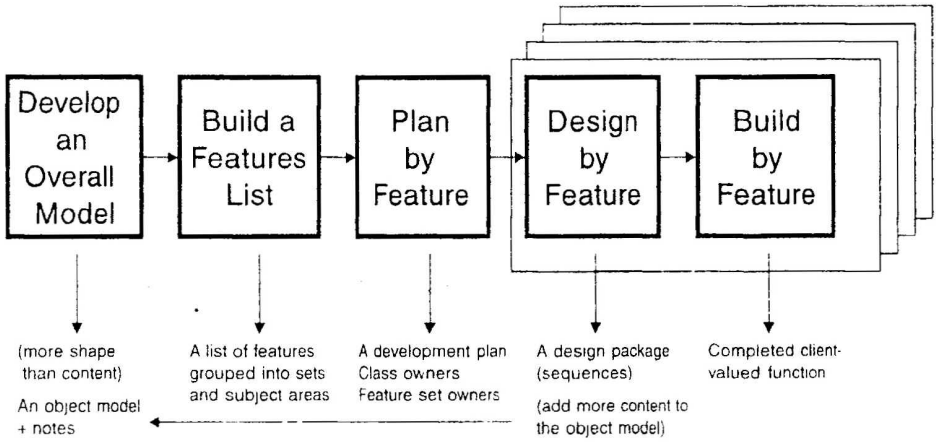

Figure 4-3
The five processes of FDD with their outputs

Mac: So FDD *consists of only five processes'. That seems simple enough, almost too simple. I have a couple of questions. The diagram shows Iteration in the last two steps but doesn't indicate iteration in the first three processes* or *in the overall process itself. So,* is *it really an iterative process?* If *so, where can* it *iterate, and how* is *that controlled?*

Steve: Within *the context of a single* FDD *project, we are iterating through the design-by-feature and build-by-feature processes* (4 *and* 5), *designing and building to completion one feature or a small group of features per iteration.* We *do a reasonable amount of work up front* in *processes* 1 *and* 3 *This ensures that when we have multiple feature teams iterating rapidly through processes* 4 *and* 5, *the output from those iterations does converge toward the desired overall result.*

The ideal is *to model the entire problem domain that lies within the scope of the project in process* 1. *However, things often fall short of ideal If, during the modeling,* it *becomes obvious that more time* is *needed* to *clarify requirements in a particular area (possibly because clients or users need management time and decisions), then we might choose to postpone the modeling of that area. Once the requirements are in better shape, the modeling team can be reformed to complete the object model in that area, the features for that area listed; and the plan, feature set assignments, and class ownership* lists *updated.* So *in that sense, yes, you can iterate through processes* 1-3 *again.*

For truly large development efforts, you often find that members of management have already decided to break the effort into several projects, some of which they might run

concurrently and some of which might be run sequentiall!:/, depending on the business priority, availability of developers, and dependencies between tne projects. Where the tnle scope of a project is discovered only during the modeling in process 1, the Project Manager needs to sit down with management to decide whether to break the project into several smaller projects, each witrl reduced scope. One criterion to use is that each project should be delivering something of real benefit to its clients.

Once a project is over, a new project can certainly select more areas of the problem domain that were outside of the scope of the original project, model those, and add new features. So *in that sense, all five* FDD *processes can be iterated.*

Mac: *What about identifying new requirements or features once you have started?* How *does that affect t{le project?*

Steve: *New requirements and features snould be controlled by a change management process, which includes reviewing tfle impact to cost, schedule, and qualit!:/, that adding new requirements to any software project can have. Then a decision can be made regarding whether to add them to the current project or leave them for a subsequent project. Let's leave discussing charlge management in detail until a little later* (Chapter 14, "Requirements Change Management").

Mac: *Okay, but I also want to know w{letner and how this applies to projects that are enhancing an existing system or customizing a core software product for a specific customer*

Steve: *Good question. Mucn* will *depend on the starting point. Adding more features to a system already built using* FDD is *simply more of the same. Repeat tne process but start witn the existing object model and features list, and extend them. Obviously, the work may also lead to some refinement of the existing object model and software. Careful documentation of any changes* is *needed, and* I'll *show you how that can be achieved without enormous overhead* (Chapter 6, "Chief Programmer Work Packages"). *Wnere software does not have an existing object model.,* it is *almost certainly going to be beneficial* to *create one. Tools such as TogetherSoft's Together Contra/Center can help reverse-engineer an existing code base. Depending on the result of the reverse engineering,* it *may be desirable to produce a more ideal object model by performing process* I *for the core system, anyway. Comparing the two models gives the Chief Architect and Project Manager a better idea of the cost of refactoring some of the existing system versus trying to bend the new features to fit within the existing model.*

Now, before you ask me any more good questions, I *really want to walk you through the five* FDD *processes in detail. However, before* I *can do that, [need to tell you a little about the format that was used when writing them down ...*

FDD in Detail

Have you ever tried to make all your developers follow a hundred-page process manual? Peter Coad tells the tale of writing a t I O-page process manual and finding that the developers only ever read the four-page summary at the back [Coad 991.

Each of the five processes in FDD is described on one or two sheets of letter-sized (A4 for those outside the United States) paper This is all the developers need and most likely all you will be able to get them to

read, anyway. It actually takes extra effort to write with simplicity, clarity, and brevity (if this book is too long, it is because we ran out of time within which to make it shorter).

Having someone who has done it before and who can answer process questions as they arise is an approach that is far more likely to see success than telling the developers to look it up in a hundred-page document. Once the Development Manager and Chief Programmers "get it," they can provide all the day-to-day help needed to guide those working with them.

The FDD Process Template

M. A Rajashima introduced the Entry-Tasks-Verification-exit (ETVX) template to Jeff De Luca, the Singapore Project Manager described in the Introduction for this book. Using this template, each FDD process is described in four sections (Table 4-1):

Table 4-1
The HVX Template

Section	Description
Entry	A brief overview of the process and a set of criteria that must be met before we can start the process
Tasks	A list of tasks to perform as part of that process
Verification	The means by which it is verified that the process has been completed satisfactorily. Is the result good enough?
Exit	A list of the outputs from the process

The tasks within each process are split into a header bar and a short text description (Figure 4-4). The header bar contains the name of the task, the role responsible for that task, and an indicator to say whether the task is optional or mandatory ("required").

Edward Tufte's magnificent book, *Envisioning Information* [Tufte], inspired the use of color or shading for layering and separation. This makes it easier to see the outline and then the detail. Also, you can run your eye down the table headings easily to see whether your role is involved. The overall result is a simple and clear structure that concisely communicates what the different people need to do in each activity.

Sometimes, organizations in industries such as pharmaceuticals, health care, or government contractors cannot accept a process written on five sheets of paper-more formality is required for legal or political

Form the Modeling Team	Project Manager	Required
The modeling team consists of permanent members from the domain and development areas, specifically, the domain experts and the chief programmers. Rotate other project staff through the modeling sessions so that everyone gets a chance to participate and to see the process in action.		
Conduct a Domain Walkthrough	**Modeling Team**	**Required**
A domain expert gives an overview of the domain area to be modeled. This should include information that is related to this domain area but not necessarily a part of its implementation.		

Figure 4-4
Task descriptions in FDD

reasons. We hope the hints and tips in the following chapters will help anyone tasked with writing the hundreds of process manual pages required to satisfy the legal regulations and organizational policies involved.

One possible place to start would be to apply the ETVX template to each of the tasks in the processes to make those more formal; the task description becomes the overview of this new subprocess. Next, define formal templates for each of the outputs and provide forms for those responsible for each task to sign off when complete.

You could also provide the processes as Web pages on an intranet and link each task heading to another page with as much justification, rationale, hints, and tips as desired. However, do not expect many of your developers to ever read it.

The rest of us who work in less regulated industries where legal and contractual accountability are less burdensome can be thankful that we deal *with* only five double-sided pages of formal process description.

ETVX, Use Cases, and Operations

The ETVX template will look vaguely familiar to those folks who use a formal template for writing use case content. Both these techniques describe what something does (algorithm), given a set of assumptions about conditions at the start (preconditions). Both describe results at the end of that activity (post conditions). Of course, many use case templates are for more complicated, but the general pattern for describing "behavior" is there.

Those practicing "program by contract" and more formal specification methods also use preconditions and post conditions. Here, the behavior is defined without specifying how it is implemented; an algorithm description or the actual source code is needed to describe the implementation.

The ideo of preconditions and post conditions seems pervasive in the description of behavior of all kinds of systems.

The verification section of the ETVX template provides something extra. It describes how to verify the quality of the result to ensure that it is "good enough." Use cases describe the actions of a deterministic computer system. ETVX is describing the actions of a software development project-a human system.. The human system has points in which value judgments are made about the quality of a design, a piece of code, a list of requirements, and a project plan.

A computer con certainly be used to help make those value judgments. For example, a set of automated audit checks and metric scripts can be run against a piece of code to highlight potential problems. The result of the checks con be used to help assess the quality of the code. However, a computer is not usually trusted to make the valuejudgments itself

---

The FDD Processes

The five process descriptions that follow have been enhanced since the publication of the coloring book [Coad 991 to make certain aspects clearer and more precise. The essential principles and practices remain exactly the same.

In subsequent chapters, we will highlight and explain the differences between the coloring book process descriptions and the improved versions published here.

FDD Process 1: Develop an Overall Model

An initial project-wide activity with domain and development members working together under the guidance of an experienced object modeler in the role of Chief Architect.

Domain Experts perform a high-level walkthrough of the scope of the system and its context. They then perform detailed domain walkthroughs for each area of the domain that is to be modeled. After each domain walkthrough, small groups are formed with a mix of domain and development staff. Each small group composes its own model in support of the domain walkthrough and presents its results for peer review and discussion One of the proposed models or a merge of the models is selected by consensus and becomes the model for that domain area. The domain area model is merged into the overall model. adjusting model shape as required

The object model is then updated iteratively with content by process 4, Design by Feature.

Entry Criteria

- Domain Experts, Chief Programmers, and the Chief Architect have been selected for the project

Tasks

Form the Modeling Team — **Project Manager** — **Required**

The modeling team consists of permanent members from the domain and development areas. specifically, the Domain Experts and the Chief Programmers Rotate other project staff through the modeling sessions so that everyone gets a chance to participate and to see the process in action.

Conduct a Domain Walkthrough — **Modeling Team** — **Required**

A Domain Expert gives an overview of the domain area to be modeled . This should include information that is related to this domain area but not necessarily a part of its implementation

Study Documents — **Modeling Team** — **Optional**

The team studies available reference or requirements documents, such as object models, functional requirements (traditional or use-case format), , and user guides .

Develop Small Group Models — **Modeling Team in Small Groups** — **Required**

Forming groups of no more than three, each small group composes a model in support of the domain area. The Chief Architect may propose a "straw man" model to facilitate the progress of the teams. Occasionally, the groups may also sketch one or more informal sequence diagrams to test the model shape.

Develop a Team Model — **Modeling Team** — **Required**

A member from each small group presents that group's proposed model for the domain area. The Chief Architect may also propose further model alternatives. The modeling team selects one of the proposed models or composes a model by merging ideas from the proposed models.

Refine the Overall Object Model — **Chief Architect, Modeling Team** — **Required**

Every so often, the team updates the overall object model with the new model shapes produced by iterations of the previous two tasks.

Write Model Notes — **Chief Architect, Chief Programmers** — **Required**

Notes on detailed or complex model shapes and on significant model alternatives are made for future reference by the project.

Verification

Internal and External Assessment — **Modeling Team, Business** — **Required**

Domain Experts actively participating in the process provide internal or self-assessment. External assessment is made on an as-needed basis by referring back to the business (users) for ratification or clarification of issues that affect the model.

Exit Criteria

To exit the process, the modeling team must produce an object model to the satisfaction of the Chief Architect. The object model consists of

- Class diagrams focusing on model shape, the classes in the domain, how are they connected to one another and under what constraints, plus any operations and attributes identified.
- Sequence diagram(s), , if any.
- Notes capturing why a particular model shape was chosen and/or what alternatives were considered

FDD Process 2: Build a Features List

An initial project-wide activity to identify all the features needed to support the requirements.

A team usually comprising just the Chief Programmers from process 1 is formed to decompose the domain functionally. Based on the partitioning of the domain by the Domain Experts in process 1, the team breaks the domain into a number of *areas (major feature sets)*. Each area is further broken into a number of *activities (feature sets)*. Each step within an activity is identified as a feature. The result is a hierarchically categorized features list.

Entry Criteria

- The modeling team has successfully completed FDD process 1, Develop an Overall Model.

Tasks

Form the Features List Team — **Project Manager, Development Manager** — **Required**

The features list team comprises the Chief Programmers from the modeling team in process 1.

Build the Features List — **Features List Team** — **Required**

Using the knowledge obtained from process 1, the features list team identifies features. The team may also use any existing reference or requirements documents, such as object models. functional requirements (traditional or use-case format): and user guides, noting the source document against any features identified in this way.

This task is a simple functional decomposition, starting with the partitioning of the domain used by the Domain Experts for their domain area walkthroughs in process 1. The domain is decomposed into *areas (major feature sets)* that comprise *activities (feature sets)* that comprise *features*, each of which represents a step in an activity.

Features are granular functions expressed in client-valued terms, using the naming template:

<action> <result><object>

For example, "calculate the total of a sale" and "calculate the total quantity sold by a retail outlet for an item description".

Features are granular. A feature should take no more than tVIOweeks to complete but not be so granular as to be at the level of simple getter and setter operations. 1Woweeks is an upper limit; most features take less than this amount of time. When a step looks larger, the team breaks the step into smaller steps that then become features

Verification

Internal and External Assessment	**Features List Team, Business**	**Required**

Modeling team members actively part:c=",~-~ :r: t r e process provide internal or self-aSSCSSIT/cnt . External assessment is mace' .:..." a r: as-,~eeded basis by referring back to the Domain Experts from the modeling tear; =~=,-,~business [users) for ratification or clarification of issues that affect the features lis:

Exit Criteria

To exit the process, the features list team must produce the features list to the satisfaction of the Project Manager and Development Manager The features list consists of:

- A list of major feature sets (areas)
- For each major feature set, a list of feature sets (activities) within that major feature set
- A list of features for each feature set (activity), , each representing a step in the activity of that feature set

FDD Process 3: Plan by Feature

An initial project-wide activity to produce the *development* plan

The Project Manager. Development Manager., and Chief Programmers plan the order that the features are to be implemented. based on feature dependencies. load across the development team. and the complexity of the features to be implemented, The main tasks in this process are not a strict sequence. Like many planning activities. they are considered together. with refinements made from one or more tasks. then considering the others again. A typical scenario is to consider the *development* sequence, then consider the assignment of feature sets to Chief Programmers and, in doing so. consider which of the key classes (only) are assigned to which developers (remembering that a Chief Programmer is also a developer), When this balance is achieved and the development sequence and assignment of feature sets to Chief Programmers is essentially completed, the class ownership is completed (beyond the key classes that were al ready considered for ownership).

Entry Criteria

- The features list team has successfully completed FDD process 2, Budd a Features List

Tasks

Form the Planning Team **Project Manager** **Required**

The planning team consists of the Project Manager, Development Manager. and Chief Programmers.

Determine the Development Sequence **Planning Team** **Required**

The planning team assigns a date (month and year only) for completion of each feature set.. The identification of the feature set and the completion date (and. thus, the development sequence) is based on:

- Dependencies between features in terms of classes *involved*
- Balancing of load across class owners
- Complexity of the features to be implemented
- Bringing forward of high-risk or complex feature sets
- Consideration of any external (visible) milestones. such as betas, previews, feedback checkpoints, and the "whole products" that satisfy, such milestones

Assign Feature Sets to Chief Programmers **Planning Team** **Required**

The planning team assigns Chief Programmers as owners of feature sets, The assignment is based on:

- Development sequence
- Dependencies between features in terms of classes involved

- Balancing of load across class owners (remembering that Chief Programmers are also class owners)
- Complexity of the features to be implemented

Assign Classes to Developers Planni~g **Team** **Required**

The planning team assigns developers as class owners. Developers own multiple classes. The assignment of classes to developers is based on:

- Balancing of load across developers
- Complexity of the classes
- Expected usage (e.g., high use) of the classes
- Development sequence

Verification

Self-ASsessment **Planning Team** **Required**

The planning is a team activity, so self-assessment is achieved by the active participation of the Project Manager and Development Manager with the Chief Programmers, who use the knowledge they gained from process I to help make better informed decisions.

Exit Criteria

To exit the process, the planning team must produce the development plan to the satisfaction of the Project Manager and Development Manager. The development plan consists of:

- Feature sets with completion dates (month and year)
- Major feature sets with completion dates (month and year) derived from the last completion date of their respective feature sets
- Chief Programmers assigned to feature sets
- The list of classes and the developers that own them (the class owner list)

FDD Process 4: Design by Feature

A per-feature activity to produce the feature(s) *design package.*

A number of features are scheduled for development by assigning them to a Chief Programmer. The Chief Programmer selects features for development from his or her "inbox" of assigned features. Operationally, it is often the case that the Chief Programmer schedules small groups of features at a time for development. He or she may choose multiple features that happen to use the same classes (therefore, developers). Such a group of features forms a *Chief Programmer Work Package.*

The Chief Programmer then forms a *feature team* by identifyjng the owners of the classes (developers) likely to be involved in the development of the selected feature(s). This team produces the detailed sequence diagram(s) for the selected feature(s). The Chief Programmer then refines the object model., based on the content of the sequence diagrarnts). The developers write class and method prologues. A design inspection is held.

Entry Criteria

- The planning team has successfully completed FDD process 3. Plan by Feature.

Tasks

Form a FeatureTeam **Chief Programmer** **Required**

The Chief Programmer identifies the classes likely to be involved in the design of this group of features. From the class ownership list, the Chief Programmer identifies the developers needed to form the feature team. As part of this step, the Chief Programmer starts a new design packagefor the feature(s) as part of the work package.

Condud a Domain Walkthroug" **Domain** Expe~ **Optional**

At the request of the Chief Programmer. a Domain Expert walks the feature team through details of algorithms. rules, formulas. and data elements neededfor the feature to be designed. This is an optional task. basedon the complexity of the feature and/or its interactions.

Study the Refe,enced Documents **Feature Team** **Optiollal**

The feature team studies the documents referenced in the features list for the feature(s) to be designed and any other pertinent documents. including any confirmation memos, screen designs, and external system interface specifications. This is an optional task, based on the complexity of the feature and/or its interactions and the existence of such documents.

Develop the Sequence Diagram(s) **Feature Team** ~Required

The feature team develops the detailed sequence diagram(s) required for each feature being designed. The team writes up and records any alternative designs, design decisions, assumptions. requirements clarifications. and notes in the design alternatives or notes section of the design package.

Refine the Object Model **Chief Programmer** **Required**

The Chief Programmer creates a *feature team area* for the feature(sj This area is either a directory on a file server, a directory on his or her personal computer (backed up by the Chief Programmer, as required). , or utilizes work area support in the project's version control system. The feature team uses the feature team area to share work in progress and make it visible among the feature team but not to the rest of the project.

The Chief Programmer refines the model to add additional classes, operations, and attributes and/or to make changes to existing classes, operations, or attributes, based on the sequence diagram(s) defined for the featurets). The associated implementation language source files are updated (either manually or automatically by some tool) in the *feature team area* The Chief Programmer creates model diagrams in a publishable format.

Write Class and Method Prologue **Feature Team** **Required**

Using the updated implementation language source files from the "Refine the Object Model" task in the feature team area, each class owner writes the class and method prologues for each item defined by the feature and sequence diagram(s). This includes parameter types, return types, exceptions, and messages.

Design Inspection **Feature Team** **Required**

The feature team conducts a design inspection, either before or after the unit test task. Other project members may participate; the Chief Programmer makes the decision to inspect within the feature team or with other project team members. On acceptance, a to-do list is created per affected class, and each team member adds his or her tasks to their to-do list.

Verification

Design Inspection **Feature Team** **Required**

A successful design inspection is the verification of the output of this process. The design inspection task is described above.

Exit Criteria

To exit the process, the feature team must produce a successfully inspected design package. The design package comprises:

- A covering memo, or paper, that integrates and describes the design package so that it stands on its own for reviewers
- The referenced requirements (if any) in the form of documents and all related conhrmation memos and supporting documentation
- The sequence diagrarnts)
- Design alternatives (if any)
- The object model with new/updated classes, methods. and attributes
- The <your tools-generated output for the class and method prologues created or modified by this design
- The to-do task-list entries for action items on affected classes for each team member

FDD Process 5: Build by Feature

A per-feature activity to produce a completed client-valued function (feature).

Working from the design package produced during the Design by Feature process, the class owners implement the items necessary for their classes to support the design for the feature(s) In the work package. The code developed is then unit-tested and code-inspected, the order of which is determined by the Chief Programmer After a successful code inspection, the code is promoted to the build.

Entry Criteria

- The feature team' has successfully completed FDD process 4, Design by Feature, for the selected feature(s). That is, the design package from process 4 has been successfully inspected.

Tasks

Implement Classes and Methods **Feature Team** **Required**

The class owners implement the items necessary to satisfy the requirements on their classes for the feature(s) in the work package This includes the development of any unit-testing code needed.

Condud a Code Inspedion **Feature Team** **Required**

The feature team conducts a code inspection, either before or after the unit test task The Chief Programmer decides whether to inspect within the feature team or with other project team members. The decision to inspect before or after unit test is also that of the Chief Programmer.

Unit Test **Feature Team** **Required**

Each class owner tests their code to ensure that all requirements on their classes for the feature(s) in the work package are satisfied. The Chief Programmer determines what., if any, feature team-level unit testing is required-in other words, what testing across the classes developed for the feature(s) is required.

Promote to the Build **Chief Programmer, Feature Team** **Required**

Classes can be promoted to the build only after a successful code inspection and unit test The Chief Programmer is the integration point for the entire feature(s) and responsible for tracking the promotion of the classes involved (either by promoting them personally or through feedback from the developers in the feature team)

Verification

Code Inspection and Unit Test	Chief Programmer, Feature Team	Required

A successful code inspection plus the successful completion of unit test is the verification of the output of this process. The code inspection and unit test tasks are described above.

Exit **Criteria**

To exit the process, the feature team must complete the development of one or more whole features (client-valued functions). To do this, it must have promoted to the build the set of new and enhanced classes that support those features, and those classes must have been successfully code-inspected and unit-tested.

Summary

FDD tackles the key core problem area in software development-that of constructing the right software correctly and on time.

FDD does enough work up front to provide a resilient conceptual framework within which to work-the domain object model (structure) and features list (requirements). The highly iterative, self-organizing, controlled chaos of the Design by Feature and Build by Feature iterations provides an agile operational framework that can quickly adapt to change.

This balanced approach advocated by FDD avoids the *analysis paralysis* often found in teams following traditional processes with long analysis phases (simply unworkable for short-duration projects with business requirements changing monthly, if not weekly). However., FDD also avoids the large amounts of unnecessary reworking of code that are almost guaranteed when a team dives straight into coding without any reasonable shared understanding of the problem to be solved.

Like other agile processes, FDD is designed to enable teams to deliver real results more quickly, without compromising quality. It is highly iterative, people-focused, and results-oriented. FDD breaks down the traditional walls separating domain and business experts/analysts from designers and irnplernenters. analysts are dragged out of their abstractions and participate first-hand with the developers and users in the construction of the system.

FDD is essentially described in five short processes, each consisting of one to two pages, following a simple ETVX template.

Other development activities, such as defining/gathering initial requirements, change management, and formal testing, provide input to or use the output of the core FDD processes.