Technical University Munich

Major Seminar

On

Feature Driven Development

Agile Techniques for Project Management
and
Software Engineering

WS 2007/08

By

Sadhna Goyal

Guide: Jennifer Schiller

Chair of Applied Software Engineering
Univ.-Prof. Bernd Brügge, Ph.D
Technical University Munich

Table of contents

List of figures

1. Introduction

In the traditional waterfall model software development approach, the whole project is divided into a number of stages: **gathering user requirements**, **design and documentation**, **development**, **testing** and **deployment**. In this approach, it assumes that each stage is 100% complete before the next stage starts. One of the main weaknesses of this approach is that design errors are often not discovered until deployment time. At this time the project is almost complete and the errors are often expensive to recover from. *"Observe that it is perhaps 100 times as costly to make a change to the requirements during system testing as it is to make the change during requirements definition*." (Fairley, R., 1985).

Agile methods try to avoid this weakness of "waterfall" by doing iterative development. Each iteration is meant to be short (1-3 weeks) and includes all of the above steps. This guarantees that design errors are discovered at the early stages of development. Feature Driven Development (FDD) is one of the agile software development methodologies that emerged in the last 10 years as an alternative to traditional "waterfall" development.

2. Birth of FDD

Jeff De Luca and Peter Coad introduced FDD in 1997. It so happened in 1997 that Jeff De Luca was the project Manager of a large software development project in Singapore. The problem domain was so complex that Jeff realized that that the task in hand could not be completed in time, with the available resources using traditional strategies of software development. He therefore with the help of Peter Coad and others discovered the modeling in color technique and the concept of feature driven development. In print this was first published in the book "*Java Modeling in Color* with UML" written by Peter Coad (Peter, et al., 1999).

3. What is FDD?

FDD is an agile, highly adaptive software development process that is

- Highly and short iterative.
- Emphasizes quality at all steps
- Delivers frequent, tangible working results at all steps
- Provides accurate and meaningful progress and status information, with the minimum of overhead and disruption for the developers.
- Is liked by client, managers and developers

4. Why do we have to use FDD?

4.1. Pains in Software Development

4.1.1. Communication

In a software development process, communication is taking place constantly at every level. In fact, no process (work) can occur without it. If we consider developers as nodes in a communication network, all potentially linked to each other by communication channels, then the number of potential communication channels increase dramatically as more number of developers are added.

4.1.2. Complexity

As the size of a software system grows, the complexity of that software system grows "*at least as fast the square of the size of the program"* [Weinberg, G., 1992] and quickly outstrips the relatively fixed capacity of a human brain. Gerald Weinberg calls this law of software development "the size / complexity dynamic".

FDD decomposes the entire problem domain into tiny problems, which can be solved in a small period of time, usually 2 weeks ➔ decomposed problems independent to each other reduces the need of communication.

"Observe that it is perhaps 100 times more costly to make a change to the requirements during system testing as it is to make the change during requirements definition" (Fairley, R., 1985).

FDD splits the project into iterations so that the distance in time between analysis and test is reduced ➔ early discovery of errors reduces the cost of fixing the errors.

4.1.3. Quality

Different persons have different perception of software quality. A user talking about the quality of a system discusses the user interface, the response time, the reliability and the ease of the use of the system. A developer talking about quality discusses elements of design; ease of maintenance and enhancement; and compliance to standards, patterns, and conventions. Software managers will look at quality in terms of ease of maintenance and enhancement, compliance to standards and conventions, and ability to deliver it on time. Project Sponsors will look at how well the system meets their business requirements. Does it allow them to meet a constantly changing business requirement and be proactive in meeting the challenges that are ever present in the marketplace? This makes necessary to view quality as a spectrum, with internal quality at one end and external quality at other end.

In FDD concept of Quality is broadened so as not just to test the code, but also include things such as coding standards, measuring audits and metrics in the code.

5. Projects and People

Projects consist of people, process, and technology, but by far the most important aspect is people. FDD defines **six key roles** and implies a number of others.

The *Project Manager* (PM) is the administrative head of the project responsible for reporting progress, managing budgets, fighting for headcount, and managing equipment, space, and resources, etc.

The *Chief Architect* (CA) is responsible for the overall design of the system. He is responsible for running the workshop design sessions where the team collaborates in the design of the system. The work requires both excellent technical and modeling skills as well as good facilitation skills. He or she steers the project through the technical obstacles confronting the project.

The *Development Manager* (DM) is responsible for leading the day-to-day development activities. In a facilitating role requiring good technical skills, the Development Manager is responsible for resolving everyday conflicts for resources when the Chief Programmers cannot do it between themselves.

The *Chief Programmers* are experienced developers who have been through the entire software development lifecycle a few times. They participate in the high-level requirements analysis and design activities of the project and are responsible for leading small teams of three to six developers through low level analysis, design and development of the new software's features.

The *Class Owners* are developers who work as members of small development teams under the guidance of a Chief Programmer to design, code, test, and document the features required by the new software system.

The *Domain Experts* are users, sponsors, business analysts, or any mix of these. They are the knowledge base that the developers rely on to enable them to deliver the correct system. Domain Experts need good verbal written and presentation skills. Their knowledge and participation are absolutely critical to the success of the system being built.

5.1. Supporting roles

The *Release Manager* ensures that the Chief Programmers report progress each week. He then reports directly to the Project Manager.

A *Language Guru* is a person who is responsible for knowing a programming language or a specific technology inside out. In projects where a programming language or technology is used for the first time, then this role is special.

The *Build Engineer* is responsible for setting up, maintaining, and running the regular build process.

The *Toolsmith* creates small development tools for the development team, test team, and data conversion team.

The *System Administrator* configures, manages, and troubleshoots any servers and network of workstations specific to the project team.

5.2. Additional roles

Testers are responsible for independently verifying that the system's functions meet the users' requirements and that the system performs those functions correctly.

Deployers convert existing data to the new formats required by the new system and work on the physical deployment of new releases of the system.

Technical Writers write and prepare online and printed user documentation.

6. FDD – Practices

FDD blends a number of Industry-recognized best practices into a coherent whole. The best practices used in FDD are:

6.1. Domain Object Modeling

It consists of building class diagrams depicting the significant types of objects within a problem domain and the relationships between them. It is a form of object decomposition. The problem is broken down into the significant objects involved. The design and implementation of each object or class identified in the model is a smaller problem to solve. When the completed classes are combined, they form the solution to the larger problem. ”Modeling in Color” is the best technique for Domain Object Modeling.

6.1.1. UML in Color

Colored UML is regular UML with color-encoded classes. The use of color allows quick understanding of the problem domain's dynamics. All the classes are divided into different

categories; each category has its own color. The auxiliary classes and interfaces are colorless. Figure 1 shows a fragment of a colored UML diagram.

Yellow: a role being played, usually by a person or an organization. For example, the user of an online auction site may play different roles: it can be a buyer, a seller or a system administrator.

Blue: a catalogue-like description. For example, an online store may have descriptions of the CD players that it sells. The description gives all the characteristics of the player, but it is not the player itself.

Green: a party, place or thing. In the previous example, the actual CD player in stock would be modeled as green. The green class usually has some identifying attributes, such as serial number, person's name etc.

Pink: a moment in time or an interval of time usually associated with some business process. For example, the fact of purchase may be shown a pink class, since it has a time of sale, which is tracked by the online store.

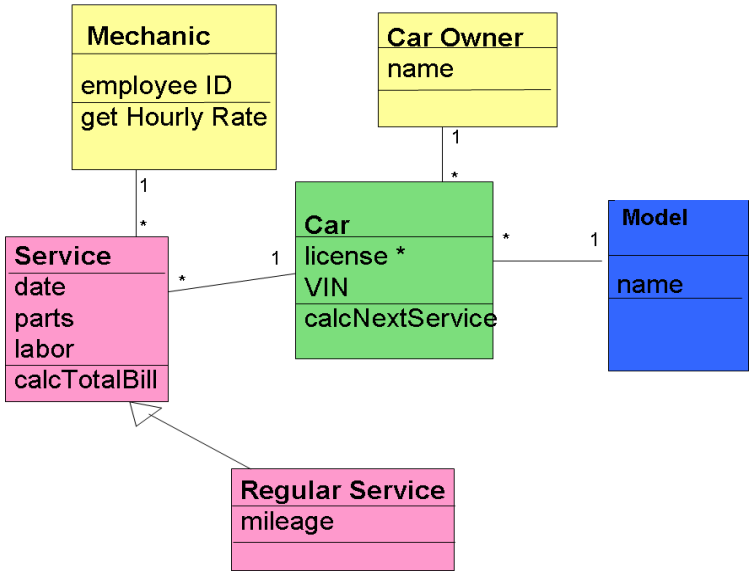

Figure 1 Example of color UML. This figure displays part of the problem domain for Feature Driven Development for a garage. The purpose of above model is to track cars in a garage. Service and Regular service class have dates , therefore they are represented by pink interval. Car is a thing so it has green color. The Model belongs to the description archetype so blue in color. A person may be car owner or a mechanic hence these classes are yellow in color. (Source: Palmer, SR., Felsing , JM.2002,p.124)

6.2. Developing by Feature

Any function that is too complex to be implemented within two weeks is further decomposed into smaller functions until each sub-problem is small enough to be called a feature. In a business system, a feature maps to a step in some activity within a business process.

By Definition: A **feature** is a small, client valued function that can be implemented in two weeks

The feature naming template is

<action>the <result><by|for |of | to|><a(n)><object>

Example of features are :

Calculate [action] the total [result] of a sale [object].

Assess the fulfilment timeliness of a sale

Calculate the total purchases by a customer

A **feature Set** is a grouping of business related features.

<action><-ing><a(n)><object>

Example : Making a product sale.

A major feature Set

<object>Management

Example: Product-Sales Management

6.3. Class (Code) Ownership

Class code ownership in a development process denotes who (person or role) is ultimately responsible for the contents of a class (piece of code). FDD uses **individual ownership** – developers are assigned ownership of a set of classes from the domain object model.

6.4. Feature Team

The implementation of a feature may involve more than one class, hence more than one Class Owner. Thus the feature Owner is supposed to have a team lead job, in which he coordinates the efforts of multiple developers.

Chief characteristics of a feature team

A feature team, due to small size of features, remains small.

As the feature team owns all the code it needs to change for that feature, there is no waiting for members of other teams to change code. So we have code ownership and a sense of collective ownership too.

Each member of a feature team contributes to the design and implementation of a feature under the guidance of a skilled, experience developer. This reduces the risk of reliance on key developers or owners of specific classes.

Class Owners may find themselves members of multiple feature teams at the same time.

Chief Programmers are also Class Owners and take part in feature teams led by other Chief Programmers.

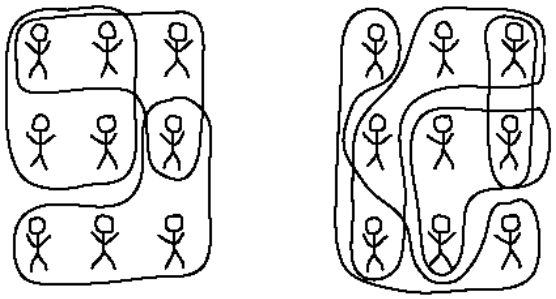

Figure 2 Feature teams: Feature teams are formed from class owners as needed. (Source: Palmer, SR., Felsing , JM.2002,p.47)

6.5. Inspections

The mix of feature team and inspection adds a new dimension to FDD. As a feature team comprises more than one member, so the fear of humiliation for a particular person is no more.Applying best known defect detection techniques and leveraging the opportunities it provides to propagate good practice, conventions, and development culture.

6.6. Regular Build Schedule

At regular intervals all the source code for the features completed is taken is taken, and the libraries and components on which it depends and the complete system is build. This ensures that there is always a demonstrable system available.

6.7. Configuration Management

This serves to identify the latest versions of completed source code files and provides historical tracking of all information artifacts in the project.

6.8. Progress Reporting

Throughout the project frequent, appropriate, and accurate progress reporting at all levels, inside and outside the project, based on completed work is done.

7. Processes

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 requirement 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. Small groups of features feature that lasts no longer than two weeks for each group and is often much shorter are taken up. FDD consists of five processes.

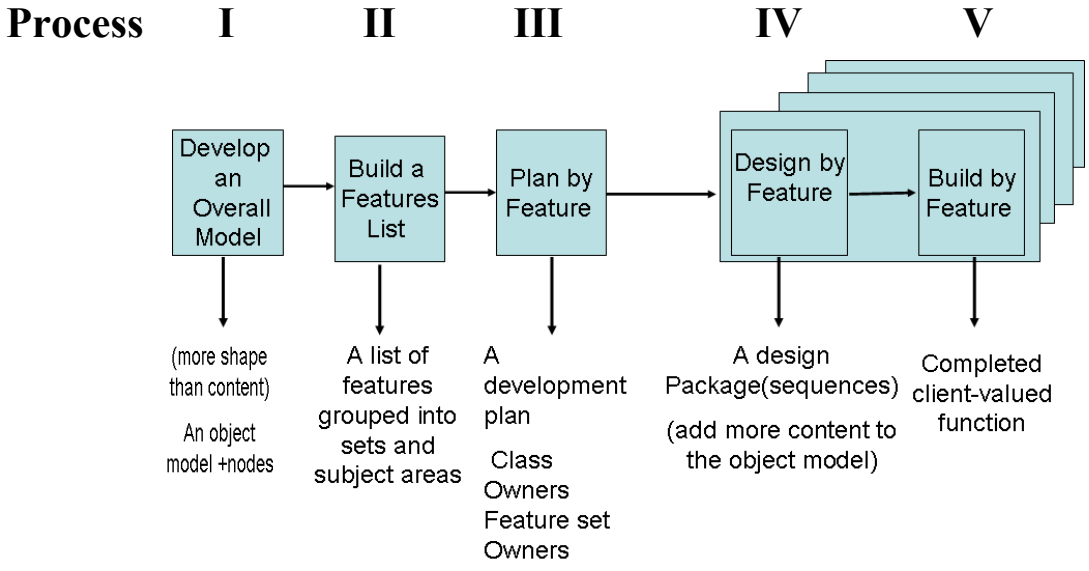

Figure 3 The five processes of FDD with their outputs (Source: Palmer, SR., Felsing , JM.2002,p.57).

7.1. Develop an Overall Model

Domain and development team members work together under the guiding hand of an experienced Object modeller (Chief Architect). Domain Members perform an initial high-level walkthrough of the scope of the system and its context. Then the domain members perform 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 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 the model shape as required. The object model is then updated iteratively with content by process IV, *Design by Feature* (see figure 3).

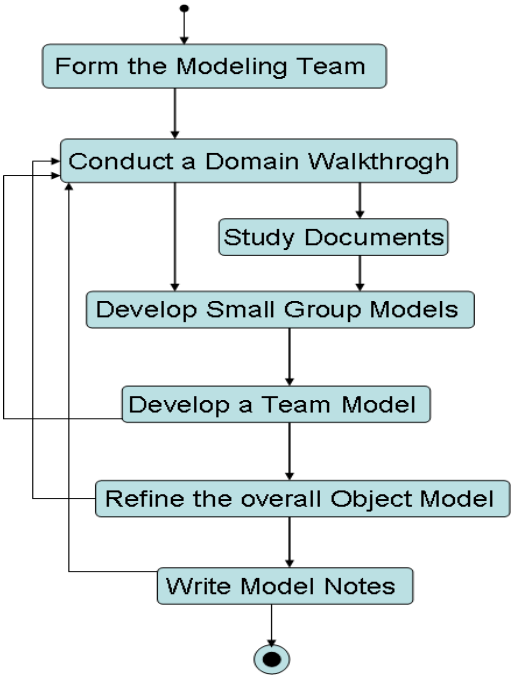

Figure 4 Flow diagram of developing an overall model (Source: Palmer, SR., Felsing , JM.2002,p.106).

7.2. Build a Features List

A team usually comprising just the Chief Programmers from process 1 is formed to decompose the domain functionality. 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.

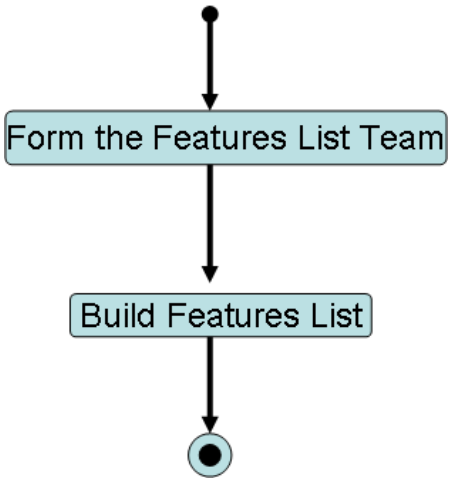

Figure 5 Flow diagram of building a feature list (Source: Palmer, SR., Felsing , JM.2002,p.135).

1	Schedule a *service* for a car
2	Edit a *customer's details* in the customer list
3	Edit the *service schedule* for a car model
4	Edit a *service description* of a service schedule
5	Edit the *task list* of a service description
6	Edit the *parts list* of a service description
7	Reserve the *list of parts* for a service
8	Send a *service reminder* to a customer
9	Edit a *service scheduled* in the workshop calendar

Figure 6 Sample feature list

7.3. Plan by Feature

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 the others are considered again. A typical scenario is to consider the development sequence, then consider the assignment of features sets to Chief Programmers, and in doing so, consider which of the key classes are assigned to which of the developers. When this balance is achieved and the development sequence and assignment of business activities to Chief Programmers is essentially completed, the class ownership is completed.

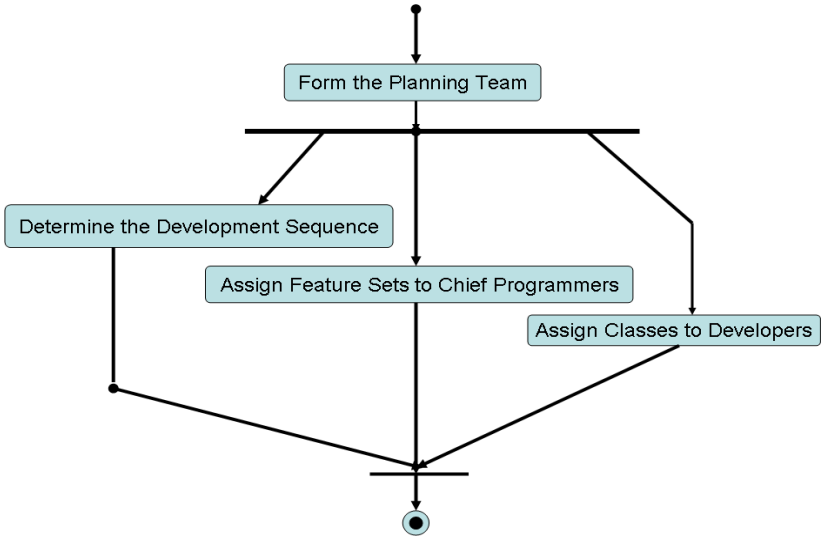

Figure 7 Flow diagram for planning by feature (Source: Palmer, SR., Felsing , JM.2002,p.146).

7.4. Design by Feature

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 group of features at a time for development. He or she may choose multiple features that happen to use the same class (hence developers). Such a group of features forms a Chief Programmer work Package. The Chief Programmer then forms a feature team by identifying the owners of the classes (developers) likely to be involved in the development of the selected feature(s). The chief Programmer then refines the object model based on the content of the sequence diagram(s). The developers write class and method prologues. A design inspection is held.

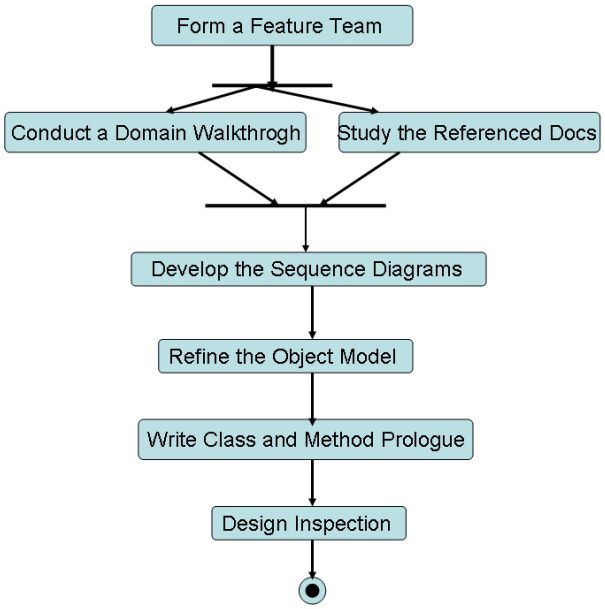

Figure 8 Flow diagram for designing by feature (Source: Palmer, SR., Felsing , JM.2002,p.160)

7.5. Build by Feature

Working from the design package produced during the Design by Feature process, the class owners implement the items necessary for their class 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 permitted to build.

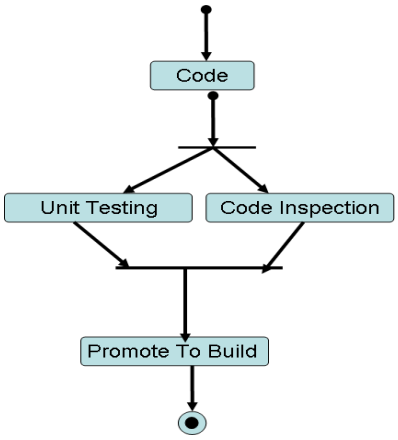

Figure 9 Flow diagram for building by feature (Source: Palmer, SR., Felsing , JM.2002,p.181).

8. Progress

8.1. Estimating Progress

FDD does not ask feature teams for a percentage of completeness. FDD tells feature teams what percentage complete they are!

8.2. Track by Feature

FDD uses six sharply defined milestones to track progress of each feature through process IV and V, Design by Feature (DBF) and Build by Feature (BBF) (Figure 10). The first three milestones are completed during the DBF process. The last three milestones are completed during the BBF process. The six milestones are completed sequentially for each feature being developed. A milestone is reported complete only when all the work for that task has been finished and verified to be so. These six milestones are as follows:

1. The Domain Walkthrough milestone is attained on completing the domain walkthrough and the optional task of studying the referenced documents.
2. The design milestone is attained on completion of the three tasks
    - Develop the sequence Diagram(s)
    - Refine the Object Model
    - Write Class and Method Prologues
3. The Design Inspection milestone is attained on successfully passing the design inspection task

4. The Code milestone is attained on completion of the implement classes and methods task.
5. The Code Inspection milestone is attained on completed of the code inspection task. This includes the completion of any modifications required by the inspection and the completion of any unit testing performed after the code inspection.
6. The Promote to Build milestone is attained when all the code for a feature has been checked into the version control system used to generate "the build".

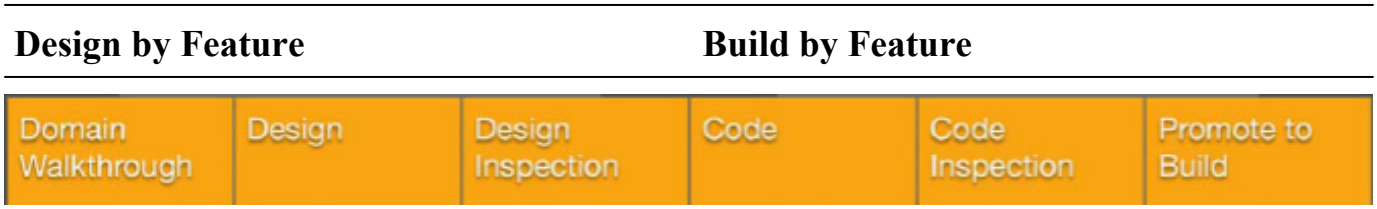

Figure 10 The six milestones of feature development. (Source: Palmer, SR., Felsing , JM.2002,p.77).

8.3. Reporting to the Chief Programmers and Project Manager

A percentage weighting is assigned to each milestone. So we can say that a feature that has reached the coding stage is 44% complete. The weighting percentages assigned to each milestone varies from situation to situation, depending upon the level of effort put into it.

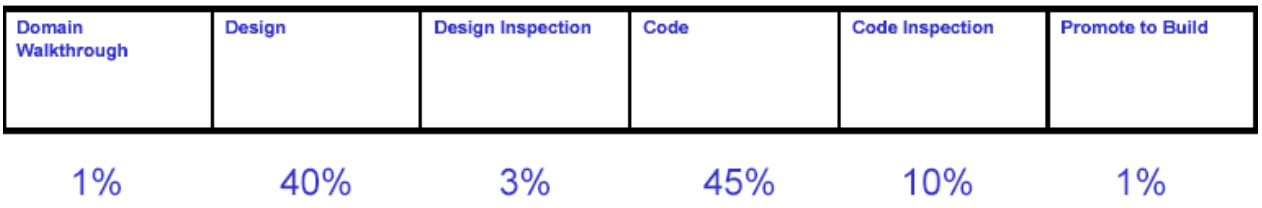

Figure 11 Percentage Weighting Assigned to Milestones(Source: Palmer, SR., Felsing , JM.2002,p.80).

Now the percentage of completeness for every feature in the feature list is calculated. Doing this for all the features in a Feature Set, gives us the completion percentage of the Feature Set. This is done for each major Feature Set and then for the whole project. In this way we can count the number of features not started, the number in progress, and the number completed for the project.

Feature Set	No of Features	No not started	No in Progress	No Completed	Percentage Completed
Scheduling a Service	19	9	8	2	27.7%
Performing a Service	15	8	7	0	30.1%
Billing a Service	6	5	0	1	16.6%
Booking in a Repair	13	2	2	9	75%
Total	53	24	17	12	38.7%

Table 1 Workshop Management Area (Source: Palmer, SR., Felsing , JM.2002,p.82)

Every week, the rate of progress is shown by plotting a graph for the number of features completed each week.

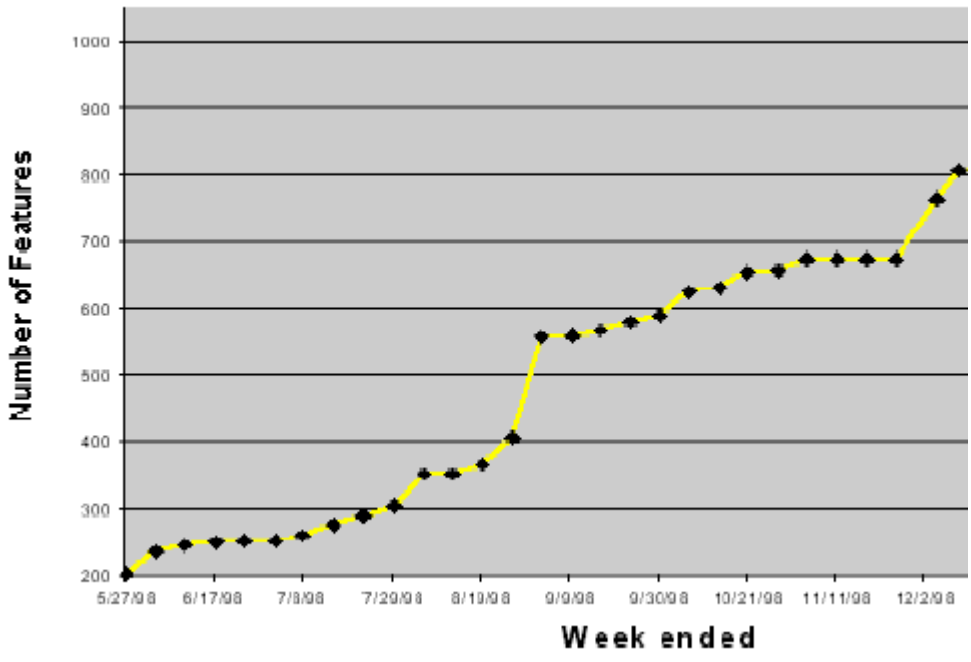

Figure 12 A graph plotted for features completed vs. Weeks elapsed

8.4. *Reporting to Sponsors and Upper Management*

Here we do not need to report for every individual feature, but just to deliver the reporting on major feature sets and their feature sets. The Figure below shows the progress of the feature set "Scheduling a Service"

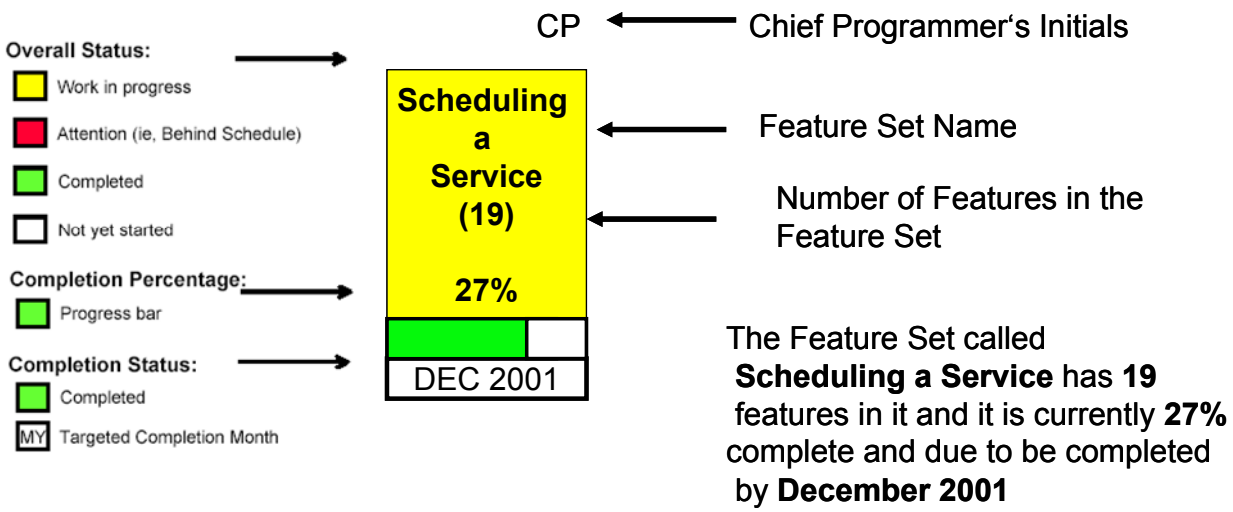

Figure 13 Progress of the Scheduling a Service feature set. (Source: Palmer, SR., Felsing , JM.2002,p.85).

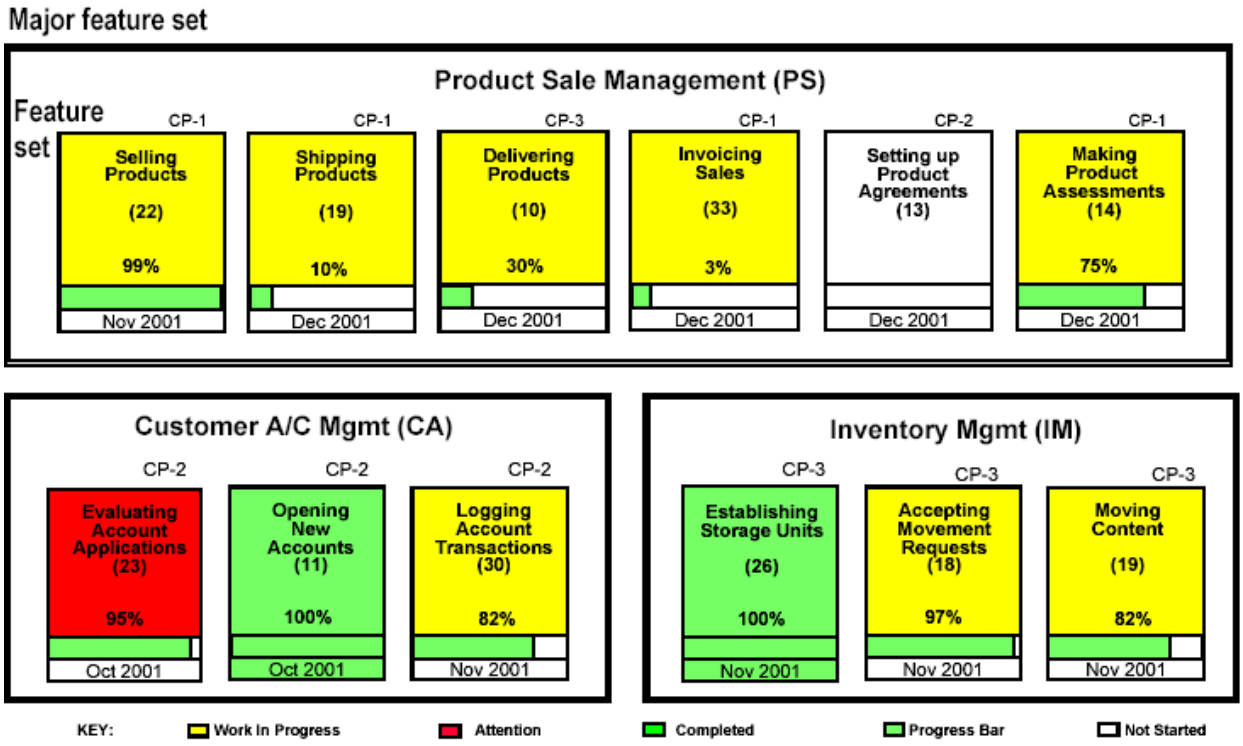

Figure 14 Feature sets progress report

In Figure 13 each feature set is represented by a rectangle divided into 3 bands: top, middle and lower. The top band is the biggest compartment and contains the name of the feature set, followed by the number of features in the feature set, followed by the percentage completed for that feature set. The middle band shows a progress bar graphically representing the percentage of completeness. The progress bar is green; the completed color. The lower band contains the planned completion date estimated in process 3 and remains white until completed, whereupon it turns green to match the other two bands.

In Figure 14, Feature Sets are arranged horizontally inside larger rectangles representing the major feature sets of the system. Each page of paper contains a number of major feature sets so that the final report consists of a few pages of colored rectangles.

Feature set	No of Features	No. not started	No. in Progress	No. Completed	No. Behind	% complete
Scheduling a Service	19	9	7	2	1	27.7%
Performing a Service	15	8	7	0	0	30.1%
Billing a Service	6	5	0	1	0	16.6%
Booking in a Repair	13	2	2	9	0	75%
Total	53	24	16	12	1	38.7%

Table 2 Tracking progress of features (behind schedule) (Source: Palmer, SR., Felsing , JM.2002,p.88).

9. Major Usage

FDD can be implemented with up to 500 developers

- More critical projects
- Bigger projects
- More novice developers
- Environments that demand Waterfall

10. Summary and Conclusion

Feature-driven development is a process for helping teams produce frequent, tangible working results. It uses very small blocks of client valued functionality, called features. It organizes those little blocks into business-related feature sets. FDD focuses developers on producing working results every two weeks. FDD is better prepared to work with team where developers' experience varies. It offers progress tracking and reporting capabilities. This comforts managers and makes it more attractive for big companies.

11. References & Links

Coad, Peter, et al. Java modeling in Color with UML.Upper Saddle River, NJ:Prentice Hall PTR, 1999.

Fairely, R. Software Engineering Concepts. New York: McGraw Hill, 1985.

Weinberg, G. Quality Software Management vols. 1-4. New York: Dorset House, 1992-1997.

Freedman, D.P., and Weinberg, GM. "Software Inspections:"An Effective Verification Process."

IEEE Software. May 31-36(1982)

Palmer, SR., Felsing, JM. "A Practical Guide to Feature Driven Development", Prentice Hall, 2002.

Internet sites

http://www.nebulon.com/

http://www.petercoad.com/

http://www.featuredrivendevelopment.com/

http://www.featuredrivendevelopment.com/certification/list

http://en.wikipedia.org/wiki/Feature_Driven_Development