chapter 5

Feature-Driven Development-Progress

Answers. How do we measure and track progress?

We plan for and track each DBF/88F *milestone. Remember that the total time from beginning to end* is *two weeks or less. So t{lese milestones are ver~ tiny- maybe 'inch pebbles' . The combination of small client-valued features and these six* DBF/BBF *milestones* is *the secret benind* FDD's *remarkable ability to track progress witn precision.*

Coad. Lefebvre. De Luca [Coad 99)

Time: A Scarce Resource

Much has been written about time being a limited resource. In software projects, this is often painfully true.

1. Software projects are often time-boxed or have required completion dates that are inflexible. In projects that have limited resources, are working with *bleeding-edge* technologies, have poorly defined requirements, or contain combinations of all three; a mandated time limit can be a major obstacle for delivering a project It may, in fact,, make the project undeliverable!
2. Process methodologies and project management practices often have administrative requirements that team members must complete as part of the work of producing a deliverable product Activities such as status reporting, filling out various forms and checkpoints, and attending numerous status meetings are a few examples of this. These activities, unless contractually required, *do not contribute* to the actual completion of the project In fact, they are responsible for consuming a valuable resource, namely, *time!* They are also usually dismissed as either inconsequential or [ust the normal "cost of doing business."

Consider a project team of 20 persons working on a three-month project. If each team member works 40 hours per week at 80% efficiency (6.4 available hours per day, assuming a 5-day week) and 4.3 weeks per month on average, this yields 1376 available hours per person for the project and 2,752 available hours in total for the project

Let's say that it takes 30 minutes per week per person to report time and status, an additional hour per week per person for status meetings, two hours per week filling out required process forms, and an additional 30 minutes per week for e-rnails. phone calls, and "visits from the boss" on project status and time reporting. That's an additional 08 hours, or 48 minutes of the available project work time per day per person (4 hours per week)-12.5% of the remaining available project time spent on "non-productive" activities. That takes 344 hours, or over 10.5 man-weeks (344/32:= 10.75) away from the total project time (the hours used are a characteristic example, based on experience, not actual figures).

The challenge, then, is to find a way to minimize activities such as time and status collection, status meetings, useless forms, and nonproductive communication activities by making the capture of that information as automatic and simple as possible.

Estimating Progress

Martin Fowler and Kent Beck, in *Planni/1g Extreme Programming* [Beck 0l], say that asking a developer for a percentage of completeness for a task generates a nearly meaningless answer.

As a developer, have you ever been in a situation where you were asked to *guesstimate* how far along on a task you were or how close you were to finishing? It was a guess because you were unsure of the actual size, scope, or complexity of the task at which you were laboring. Also, the time required to produce an accurate answer would significantly delay the completion of the task.

As a team lead or Project Manager, you will probably have encountered many developers who have dived into coding without thinking enough first. These developers are often 90% complete in a matter of days. 9?% complete in a month, 99% complete in six months; then they depart for a better job in 12 months, leaving the work "99.9% complete."

Feature-Driven Development (FDD) does not ask feature teams for a percentage of completeness. FDD *tells* feature teams what percentage complete they are'.

FDD uses the percentage of completeness of each feature to produce summary progress reports accurately to all levels of management within and outside the project team. The coloring book [Coad 991 suggests one particular way of spatially organizing progress reports for upper management., sponsors, and clients that uses a simple color-coding scheme to communicate more effectively We discuss these in more detail and add suggestions for additional reports to the various interested parties within the project

It should also be said that the FDD processes do not dictate what reports are produced or their format. It merely enables the reports to be produced accurately and with a minimum of effort.

Mac: *What you're saying is that FDD provides accurate reporting as a result of using the process and recording progress?*

Steve: *Yes! That's one of the biggest issues encountered with many of the other software development processes that I've used. The reporting of status is either so inaccurate as to be useless or misleading, or it takes so much effort to do on the part of the developer that it never gets done-and actually takes valuable time away from the project.*

Track by Feature

We start by defining six milestones for each feature. We iterate through the feature list, performing the Design by Feature (DBF)/Build by Feature (BBF) processes for each feature. The first three milestones are completed during the Design by Feature process. The last three milestones are completed during the Build by Feature process. The six milestones are completed sequentially for each feature being developed (Table 5-1).

Table 5-1
The Six Milestones of Feature Development

Design by Feature			Build by Feature		
Domain Walkthrough	Design	Design Inspection	Code	Code Inspection	Promote to Build

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(sJ,
   - 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 and, if unit testing is being done before code inspection, on completion of the unit test task.
5. The *Code Inspection* milestone is attained on completion 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."

Consider a project team of 20 persons working on a three-month project If each team member works 40 hours per week at 80% efficiency (64 available hours per day, assuming a 5-day week) and 4.3 weeks per month on average, this yields 137.6 available hours per person for the project and 2,752 available hours in total for the project

Let's say that it takes 30 minutes per week per person to report time and status. an additional hour per week per person for status meetings. two hours per week filling out required process forms, and an additional 30 minutes per week for e-rnails, phone calls, and "visits from the boss" on project status and time reporting. That's an additional 0.8 hours, or 48 minutes of the available project work time per day per person (4 hours per week)-I 2.5% of the remaining available projecttime spent on "non-productive" activities. That takes 344 hours, or over 10.5 man-weeks (344/32 = 10.75) away from the total project time (the hours used are a characteristic example, based on experience, not actual figures).

The challenge. then. is to find a way to minimize activities such as time and status collection, status meetings, useless forms, and nonproductive communication activities by making the capture of that information as automatic and simple as possible.

Estimating Progress

Martin Fowler and Kent Beck, in *Planning Extreme Programming* [Beck 01), say that asking a developer for a percentage of completeness for a task generates a nearly meaningless answer.

As a developer, have you ever been in a situation where you were asked to *guesstimate* how far along on a task you were or how close you were to finishing? Jt was a guess because you were unsure of the actual size, scope, or complexity of the task at which you were laboring. Also, the time required to produce an accurate answer would significantly delay the completion of the task.

As a team lead or Project Manager, you **will** probably have encountered many developers who have dived into coding without thinking enough first. These developers are often 90% complete in a matter of days, 95% complete in a month, 99% complete in six months; then they depart for a better job in 12 months, leaving the work "999% complete".

Feature-Driven Development (FDD) does not ask feature teams for a percentage of completeness. FDD *tells* feature teams what percentage complete they are:

FDD uses the percentage of completeness of each feature to produce summary progress reports accurately to all levels of management within and outside the project team. The coloring book [Coad 991 suggests one particular way of spatially organizing progress reports for upper management,, sponsors, and clients that uses a simple color-coding scheme to communicate more effectively We discuss these in more detail and add suggestions for additional reports to the various interested parties within the project.

It should also be said that the FDD processes do not dictate what reports are produced or their format.. It merely enables the reports to be produced accurately and with a minimum of effort.

Mac: What you're saying is that FDD provides accurate reporting as a result of using the process and recording progress?

Steve: Yes'. That's one of the biggest issues encountered with many of the other software development processes that I've used. The reporting of status is either so inaccurate as to be useless or misleading, or it takes so much effort to do on the part of the developer that it never gets done-and actually takes valuable time away from the project.

Track by Feature

We start by defining six milestones for each feature. We iterate through the feature list, performing the Design by Feature (DBF)/Build by Feature (BBF) processes for each feature. The first three milestones are completed during the Design by Feature process. The last three milestones are completed during the Build by Feature process. The six milestones are completed sequentially for each feature being developed (Table 5-1).

Table 5-1
The Six Milestones of Feature Development

Design by Feature			Build by Feature		
Domain Walkthrough	Design	Design Inspection	Code	Code Inspection	Promote to Build

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 and. if unit testing is being done before code inspection, on completion of the unit test task.
5. The *Code Inspection* milestone is attained on completion 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."

The key point here is that a milestone is reported complete *only* when all work for that task has been finished and verified to be so. No estimation is required or allowed'.

Each of these milestones is, therefore, very explicitly defined to satisfy Fred Brooks' observation that "rarely will a man lie about milestone progress, if the milestone is so sharp that he can't deceive himself" [Brooks]. This is a deep and really key point about FDD and why it is so effective and operational, , the milestones are the key to the accurate tracking.

Of course, we need to ask the Chief Programmers to indicate when each of the milestones is reached. A tick in a box allows us to determine where we are currently; having the Chief Programmers record the actual date a milestone is reached enables us to examine the progress at specific points in time and to plot trends in progress over time.

Mac: So, *by using well-defined milestones for the Design by Feature and Build by Feature processes of* FDD, *you're saying trwt we can accurately report actual progress and eliminate the guesswork.*

Steve: *Exactly'.*

Mac: *How do we capture this information? You mentioned that the Chief Programmers check off the milestone in a box. Do we need a special software package to do this?.*

Steve: No. *In fact., haVing to enter information into an online form could slow trlings down. I guess a laptop* or *(land-held device migflt work but J've found that the best way to do* tftis is *for tl1e Chief Programmers to be given printouts of t(1e worh packages* (see Chapter 6. "Chief Programmer Work Packages") *t(wt they are workIng on and* to *have tf1em scribble t(1e dates in the boxes as they go along. At regular intervals most uSlwlly once a week, t(Je Chief Programmers meet to present their scribblings to the release manager, who oversees the actual data entr!:! and processing.*

Reporting to the Development Team

By tracking the dates that each of the milestones is attained for each feature, we can produce a large report, listing every feature, which Chief Programmer owns each feature, and the dates of each milestone achievement so far. This can be posted on a suitable wall in the development team area, in some place visible and accessible to the project team. Near the coffee machine or water cooler is often a good place

We can also use a simple coloring scheme to show the status of each feature. For example, we can leave a feature white if it has not been started, color it yellow if it is in progress, and green when it is complete. This simple coloring scheme allows people to stand back from the wall and get a good visual feel for the overall status of the project. then walk up to the wall to zoom in on particular areas in more detail Unfortunately, it does not work quite so well in grayscale. Thankfully a cheap, color inkjet printer does more than an adequate job Table 5-2 is an example of tracking milestones.

Table 5–2
Tracking Feature Development

#	Feature Name	Chief Prog.	Domain Walkthru	Design	Design Inspection	Code	Code Inspection	Promote to Build
1	Schedule a *service* for a car	srp	12/10	12/10	15/10	17/10	19/10	22/10
2	Edit a *customer's details* in the customer list	srp	12/10	12/10	15/10	17/10	19/10	22/10
3	Edit the *service schedule* for a car model	srp	22/10	22/10	24/10	29/10	30/10	31/10
4	Edit a *service descrip* of a service schedule	srp	22/10	22/10	24/10	29/10	30/10	31/10
5	Edit the *task list* of a service description	srp	22/10	22/10	24/10	29/10	30/10	31/10
6	Edit the *parts list* of a service description	srp	22/10	22/10	24/10	29/10	30/10	31/10
7	Reserve the *list of parts* for a service	srp	22/10	22/10	24/10	29/10	30/10	31/10
8	Send a *service reminder* to a customer	srp	22/10	22/10	24/10	29/10	30/10	31/10
9	Edit a *service* in the workshop calendar	srp	22/10	22/10	24/10	29/10	30/10	31/10

Mac: *If we have daily builds, do we need to print this wall chart every day? That would be a full-time job.*

Steve: *No, we usually print it only after a release meeting, and that is usually held only once or, at most, twice a week. This is usually more than frequently enough for any project of any significant size. For a small project, we could have a mini-release meeting at the end of each day and update the wall chart accordingly.*

Mac: *If we had an online system that Chief Programmers could use to enter dates as they went along, then we could have an up-to-the-minute view of the feature chart on-line.*

Steve: *Yes, there is a great deal of fun that could be had building useful visualizations of the data and a virtual representation of the big wall chart. However, we do not want to get carried away and spend more time than necessary producing wonderful charts and reports when we could be adding function to the requested system, instead.* Com-*man sense needs to prevail*

Mac: *Agreed! We are being paid to produce results, not status reports. It might make a nice project sometime, though.*

Reporting to the Chief Programmers and Project Manager

With our sharp milestones defined, we next assign a percentage weighting to each of them (Table 5-3).

Now we can say that a feature that has reached the coding stage is 1 + 40 + 3 = 44% complete. Notice that we *count only the milestones that have been reached* and do not take any work in progress into account. In this example, we may be 90% through the coding tasks but it has not been completed, so it is not counted.

Table 5-3
Percentage Weighting Assigned to Milestones

Design by Feature			Build by Feature		
Domain Walkthrough	Design	Design Inspection	Code	Code Inspection	Promote to Build
1%	40%	3%	45%	10%	1%

This might seem strange at first glance, but the error is small because each feature is granular and represents no more than two weeks of time. In this respect, the reporting lags a little behind the work (usually during design or code milestones), but does err on the conservative side. Using only completed tasks to report status means that we are measuring accomplished goals, rather than estimated progress.

"Mister Scot, have you always multiplied your repair estimates by a factor of four?" '

"Certainly, Sir. *How else can* I *keep my reputation as a miracle worker?" '*
Captain James T Kirk and Chief Engineer Montgomery Scot,
Star Trek III, *The Search for Spack,* Paramount Picture Corporation 1984

The weighting percentages assigned to each milestone in Table 5-3 are a starting point. It is important that the percentages reflect reality as closely as possible. If, for instance, we know that domain walkthroughs in our organization take 5% of the effort, not 1%, that design takes only 20%, and that design inspections really use 10%, not 3%, of the effort, then our weightings should reflect this. This is true even if the weightings in our organization differ from industry averages. The whole point of doing this is to produce accurate reports and not kid ourselves or management about progress.

Mac: *We've never done projects this way before.* I *don't have a feel for how dose those percentages are to the actual performance of our team and organization.* How *do we deal with that?.*

Steve: *The percentages* I *presented were really a starting point and can &e used for the first couple of projects* or *su&projects. Then, as the actual performance information* is *captured, you can adjust the percentages for any follow-on projects. If you do adjust them, you should record that information as to when and why, and make that information available to the entire organization.*

The weightings should not be changed while features are in the process of being developed without the knowledge of the sponsor, client, and upper management because a shift in the weightings could lead to lower overall completeness percentage (as we will see later), and that would take some explaining at the next progress review with that audience. In a large project, there might be planned review points, with big milestones marking the end of a specific time box or the completion of a specified number of features. These make good points to review progress and agree on an adjustment of weightings with sponsors, clients, and upper management.

We can now calculate the percentage of completeness for every feature in the feature list. We can easily roll up the percentage figures for each feature in a feature set to calculate the percentage of completeness for each feature set. We can do the same for each major feature set and for the whole project. We can also easily count the number of features not started, the number in progress, and the number completed for the project or a particular time box within the project and produce a report such as the one shown in Figure 5-1.

Workshop Management Feature Area

Feature Set	Number of Features	Number Not Started	Number in Progress	Number Completed	Percentage Complete
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%**

Car Sales Management Feature Area

Feature Set	Number of Features	Number Not Started	Number in Progress	Number Completed	Percentage Complete
Ordering a New Car	16	9	4	3	30%
Selling a Forecourt Car	7	0	0	7	100%
Selling an Approved Car	7	0	0	7	100%
Trading in an Old Car	15	5	0	10	66.7%
Arranging Financing	13	1	3	9	78%
Total	**58**	**15**	**7**	**36**	**63.9%**
...	...	...	...	...	
Complete Project Total	**164**	**64**	**32**	**58**	**45%**

Figure 5-1 .
Progress totals summary report

If we generate the data for this report on a regular basis, for example, every Wednesday afternoon, we can plot trends over time to show rates of progress. For example, we can plot and graph the number of features completed each week (Figure 5-2).

This is very useful for the Chief Programmers and Project Managers to determine whether the underlying rate of feature completion is increasing, decreasing, or stable.

This sort of graph can also be used to demonstrate the positive impact of discovering and purchasing a useful third-party component; the line suddenly shoots up steeply It can also be useful in communicating to upper management, sponsors, and clients the negative impact of doing things such as a major formal system demonstration; the line goes almost flat for three weeks as the team prepares for the demo, does the demo, and recovers from it

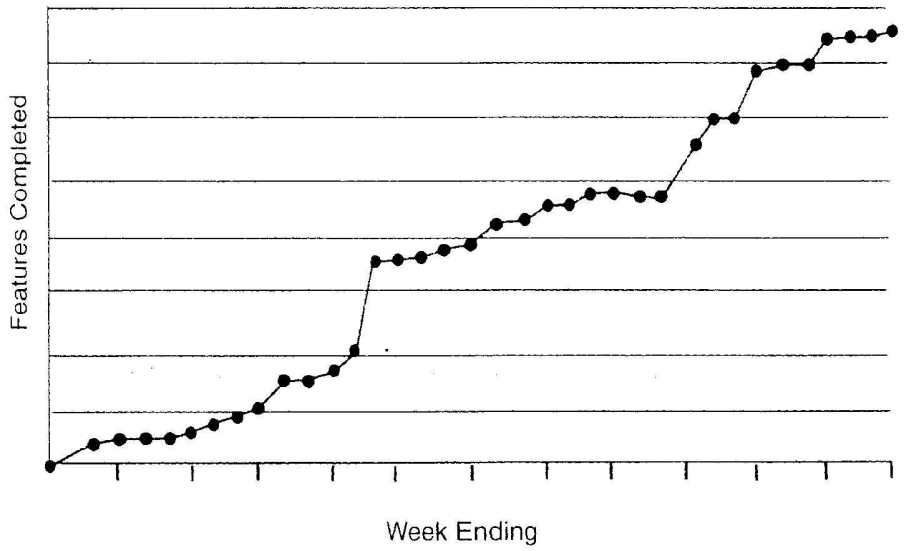

Figure 5-2
Features completed vs weeks elapsed

Mac: *Hey, Steve. Why does the cfwrt from your previous project show a large iump in the number of features completed? What happened there?* Did *you suddenly all become flyper productive?*

Steve: *WelL* sort *of We found, bought, and integrated a third-party component trwt provided a large number of spreadsheet-like features that we needed.* A *large number of features ticked off the list, and it was a great illustration to show management and developers the benefits of reusable components*

Mac: *Very cool! And the flat bits? What happened there?* How *come the productivity dropped?*

Steve: *The worse flatline, where we completed little over a period of trnee weeks, was when management suddenly decided it would be good to do a major formal system demonstration to show everyone the progress* so *far* It *meant that tile project leadership and some of the Cl1lef Programmers were workif1g on the logistics of this formal demonstration instead of resolving issues and ensuring that features were properly completed*

Again, being able to show management the flatline helped to persuade them to give us more notice the next time. Work didn't stop, it just didn't get finished; once the demo was over, the completion rate picked up again quickly.

Mac: *What about the smaller dips? I guess those could show people on vacation or sick.*

Steve: *Could be. I honestly do not remember. There were public holidays and other scheduled major progress review points but I couldn't match them up with dips in the graph with any certainty.*

Mac: *Might be worth annotating the graph with significant events next time to help show exactly why output dropped.*

Steve: *Yes, good idea. I'll remember to do that on this project.*

All sorts of graphs can be plotted and mathematics applied to produce statistics and metrics, none of which would be possible without the data provided by tracking the six milestones at the level of each feature.

Reporting to Sponsors and Upper Management

When it comes to preparing progress reports for management, clients, and sponsors, we do not need to report on every individual feature; reporting on major feature sets and their feature sets is more than good enough at this level.

We want to communicate the status of the project clearly and concisely. A gentle graphical representation of each feature set plus the use of color again can aid communication enormously. We again use yellow to mean work in progress, green to mean completed, and white to mean not started.

We also add red, indicating that something needs attention (in other words, something is slipping behind schedule). We can do this because we have the planned dates that were estimated in process 3 for each feature set. Therefore, if a feature set is not completed by the estimated planned date, it shows up in red.

Figure 5-3 shows each feature set represented by a rectangle divided into three 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. This compartment is set to the background color appropriate for the status of the 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.

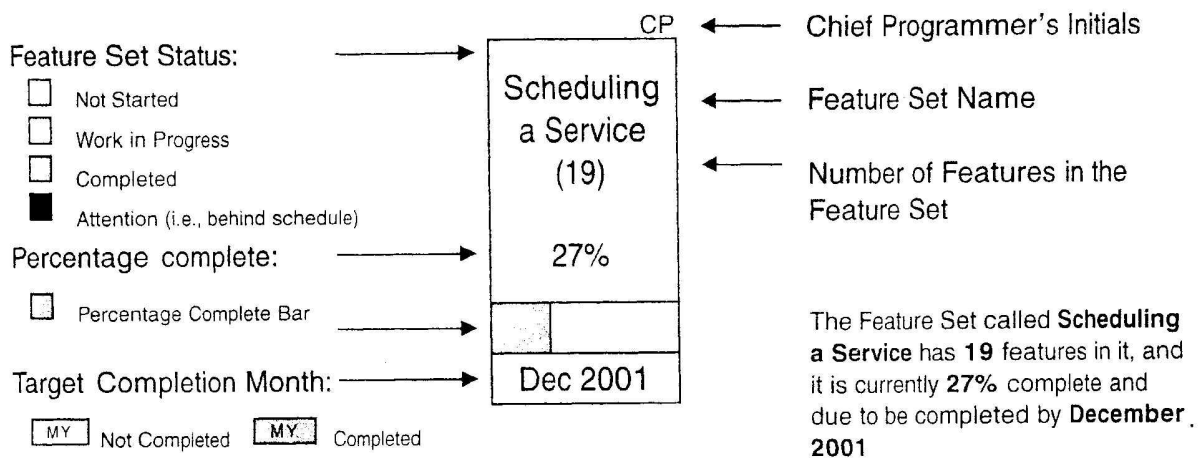

Figure 5-3
Progress of the Scheduling a Service feature set

lust to add a little bit of incentive to the Chief Programmers, although they are not normally the type of people who need it,, their initials are printed at the top right-hand corner of each of the feature sets they own.

Feature sets are arranged horizontally inside larger rectangles representing the major feature sets of the system (Figure 5-4). Each page of paper contains a number of major feature sets so that the final report consists of a few pages of colored rectangles.

Keep the layout the same for each report sent to upper management, sponsors, and clients so that it is easy to look back at previous reports and see (hopefully) green appearing in increasing amounts, red appearing, then disappearing, and a similar amount of yellow throughout the middle part of the project..

Clients, upper management, and sponsors soon learn to ask why a feature set is red and why another feature set has been yellow for a long time and still has not turned green. They should also be concerned if they see the Chief Programmer initials changing frequently in the report. This could indicate a number of things, nearly all of them negative, including:

- High turnover in the team personnel
- Frequent reassignments, hiding a Chief Programmer who is not delivering
- A Project Manager desperately trying to control a runaway project

Mac: *You could helve some fun elnimating the changes in color over time in this report.*

Steve: *It isn't {1igh on my priority list.* ©

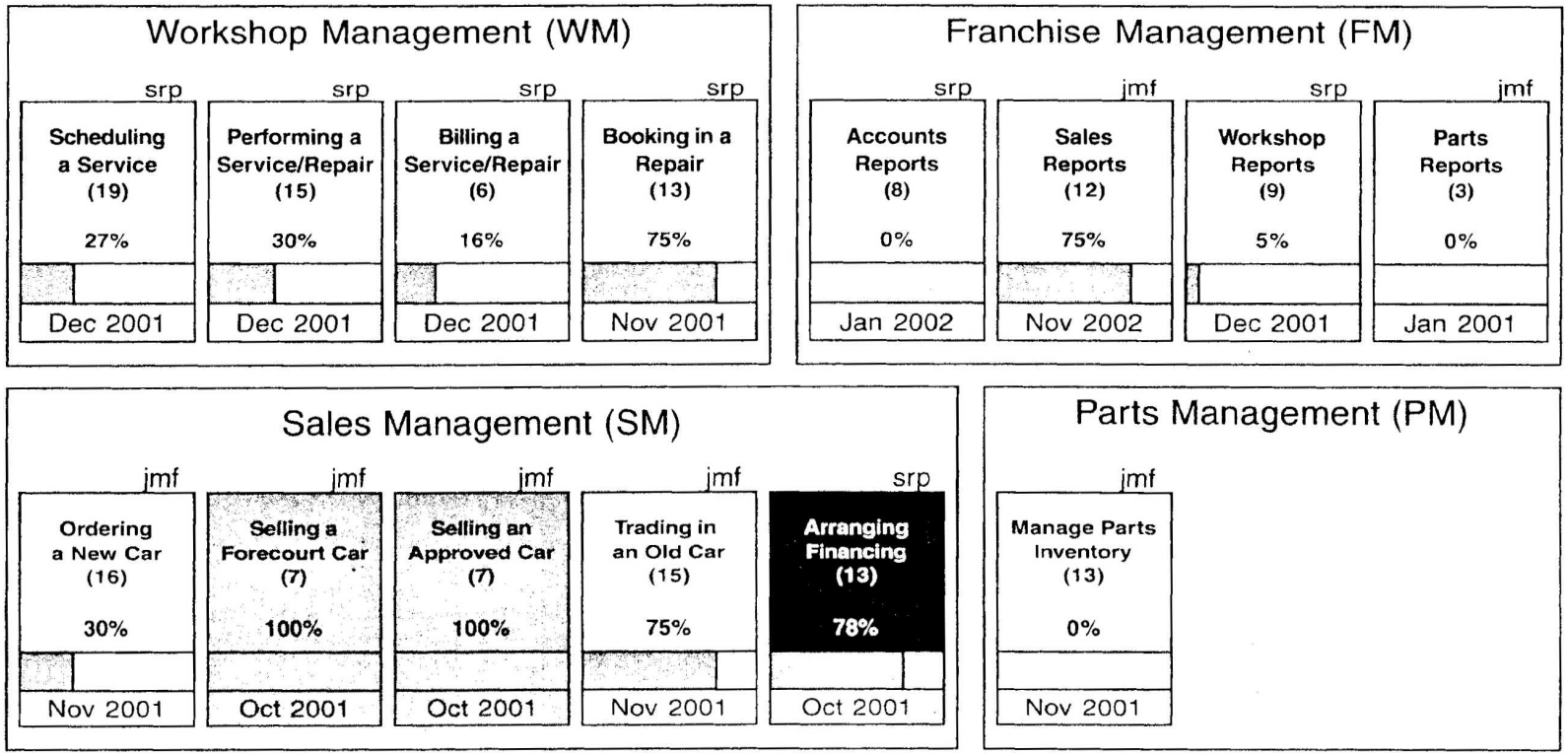

Figure 5–4
Feature sets progress report

Mac: *I really like this. It seems to be mud1 easier than trying to explain a detailed Gantt chart to a group of senior business managers Gantt charts manage to cram an enormous amow·1t. of information into one diagram but they can get very' complicated.*

Steve: *And they can be abused. I've seen one project present a Gantt chart that was pages of tasks, all with the sa me start and end dates-the start and end of the project.. It was just page after page of bars going from one end of the page to another. As far as I could tell, the project lead was essentially saying, "We are not sure but we think we are going to be very busy doing a lot of work for the whole duration of the project.".*

Mac: *That is extreme! I've seen Gantt charts used very' well, and they can be simple enough when done at a high level.. There are a couple of things shown on a Gantt chart that are not shown in your parking lot chart. . The first is start dates and the second is dependencies.*

Steve: *Correct. . We capture only completion dates and not start dates or actual hours worked on a particular task. You could extend the scheme to capture these You might want this if you usually have people working on multiple projects at a time and have to account for the time spent on each. I'd prefer to keep it as simple as possible if we can.*

Mac: *Yes. Knowing what's completed and what is not is probably enough. Another thing to bear in mind, though, is that people do not like change. Suddenly turning up at a major project status meeting with a new style of report may not go down too well. We need to be gentle! [']II find out the type of report that upper management, sponsors, and clients usually expect We can then decide whether that report communicates progress well enough and can be easily produced with minimum disruption to the developers. If it can, there seems to be little reason to change it.. If it does need changing, we'll take both to the first status meeting and try to get management to agree to adopt the new format. .*

Steve: *Fair enough . It will not affect us on tr1is project but if you ever try to use the color-coding abroad sometime, it is worth chec~ing that the colors are still appropriate Different colors have different symbolism in different countries. You have to be careful and not overly rigid about the color scheme. It is far better to find the appropriate color scheme for the culture and use that..*

Chief Programmer Plans

So far, we have looked at tracking progress. In the last section, we also compared progress against a plan and usedthat information to highlight problem areas to upper management, sponsors, and clients . Can we do the same at the Chief Programmer level?: The answer is yes.

We can ask the Chief Programmers to provide plan dates for each of the six DBF/BBF milestones for the features they are about to start working on. Then we can compare the actual dates with the planned dates and color the features red that are behind schedule In this way, we are treating each DBF/BBF iteration as some sort of micro project. , with the Chief Programmer as both the architect and Project Manager

We can now add another column to our project report (Figure 5-5)...

... and add planned dates to the wall chart, coloring behind-schedule features accordingly (Table 5-4).

Workshop Management Feature Area

Feature Set	Number of Features	Number Not Started	Number in Progress	Number Behind	Number Completed	Percentage Complete
Scheduling a Service	19	9	7	1	2	27.7%
Performing a Service	15	8	7	0	0	30.1%
Billing a Service	6	5	0	0	1	16.6%
Booking in a Repair	13	2	2	0	9	75%
Total	**53**	**24**	**17**	**2**	**12**	**38.7%**

Car Sales Management Feature Area

Feature Set	Number of Features	Number Not Started	Number in Progress	Number Behind	Number Completed	Percentage Complete
Ordering a New Car	16	9	2	0	3	30%
Selling a Forecourt Car	7	0	0	0	7	100%
Selling an Approved Car	7	0	0	0	7	100%
Trading in an Old Car	15	5	0	0	10	66.7%
Arranging Financing	13	1	1	2	9	78%
Total	**58**	**15**	**7**	**2**	**36**	**63.9%**
...	...	...	...		...	
Complete Project Total	**164**	**64**	**24**	**8**	**58**	**45%**

Figure 5-5
Tracking progress of features (column added for features behind schedule).

Table 5–4
Tracking Behind-Schedule Features

#	Feature Name	Chief Prog.	Domain Walkthrough		Design		Design Inspection		Code		Code Inspection		Promote to Build	
			Plan	*Actual*	*Plan*	*Actual*	*Plan*	*Actual*	*Plan*	*Actual*	*Plan*	*Actual*	*Plan*	*Actual*
1	Schedule a *service* for a car	srp	12/10	12/10	12/10	12/10	15/10	15/10	17/10	17/10	19/10	19/10	22/10	22/10
2	Edit a *customer's details* in the customer list	srp	12/10	12/10	12/10	12/10	15/10	15/10	17/10	17/10	19/10	19/10	22/10	22/10
3	Edit the *service schedule* for a car model	srp	22/10	22/10	22/10	23/10	24/10	24/10	29/10		30/10		31/10	
4	Edit a *service description* of a service schedule	srp	22/10	22/10	22/10	23/10	24/10	24/10	29/10		30/10		31/10	
5	Edit the *task list* of a service description	srp	22/10	22/10	22/10	23/10	24/10	24/10	29/10		30/10		31/10	
6	Edit the *parts list* of a service description	srp	22/10	22/10	22/10	23/10	24/10	24/10	29/10		30/10		31/10	
7	Reserve the *list of parts* for a service	srp	22/10	22/10	22/10	23/10	24/10	24/10	29/10		30/10		31/10	
8	Send a *service reminder* to a customer	srp	22/10	22/10	22/10	23/10	24/10	24/10	29/10		30/10		31/10	
9	Edit a *service scheduled* in the workshop calendar	srp	22/10	22/10	22/10	23/10	24/10	24/10	29/10		30/10		31/10	
...	...													
20	Make a *mechanic assignment* for a service	srp												
21	Record a *service performed* for a car	srp	25/10	25/10	26/10	26/10	31/10		2/11		5/11		7/11	
22	Record *any ad hoc notes* for a service	srp	25/10	25/10	26/10	26/10	31/10		2/11		5/11		7/11	
23	Record a *list of parts* used for a service	srp	25/10	25/10	26/10	26/10	31/10		2/11		5/11		7/11	
24	Record a *total of labor time* for a service	srp	25/10	25/10	26/10	26/10	31/10		2/11		5/11		7/11	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
35	Calculate a *total cost of parts used* for a service	srp												
36	Calculate a *total cost of labor* for a service	srp												
37	Calculate a *tax total* for a service	srp												
38	Generate an *invoice* for a service	srp												
39	Process a *payment received* for a service	srp												
40	Generate a *receipt* for a payment	srp	1/10	1/10	2/10	2/10	4/10	4/10	8/10	8/10	9/10	10/10	9/10	10/10
41	Edit *problem details* for a repair	srp	10/10	10/10	10/10	11/10	15/10	15/10	19/10	19/10	22/10	22/10	24/10	24/10
42	Edit a *repair scheduled* in the workshop calendar	srp	10/10	10/10	10/10	11/10	15/10	15/10	19/10	19/10	22/10	22/10	24/10	24/10
43	Add the *details of a car* to the list of cars	srp	10/10	10/10	10/10	11/10	15/10	15/10	19/10	19/10	22/10	22/10	24/10	24/10
43	Add the *details of a car* to a repair	srp	10/10	10/10	10/10	11/10	15/10	15/10	19/10	19/10	22/10	22/10	24/10	24/10

(continued)

Table 5–4
Tracking Behind-Schedule Features

#	Feature Name	Chief Prog.	Domain Walkthrough		Design		Design Inspection		Code		Code Inspection		Promote to Build	
			Plan	*Actual*	*Plan*	*Actual*	*Plan*	*Actual*	*Plan*	*Actual*	*Plan*	*Actual*	*Plan*	*Actual*
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
54	Update the *model catalogue* from the makers info	jmf												
55	Create a *new order* for a car model	jmf	1/10	1/10	2/10	3/10	4/10	5/10	9/10	10/10	11/10	11/10	12/10	12/10
56	Edit the *optional items* for a new car order	jmf	1/10	1/10	2/10	3/10	4/10	5/10	9/10	9/10	11/10	11/10	12/10	12/10
57	Cancel a *car order* for a customer	jmf	1/10	1/10	2/10	3/10	4/10	5/10	9/10	9/10	11/10	11/10	12/10	12/10
58	Submit a *new car order* to the manufacturer	jmf	15/10	18/10	19/10	19/10	22/10	22/10	26/10		29/10		31/10	
59	Print an *order confirmation* for the customer	jmf	15/10	18/10	19/10	19/10	22/10	22/10	26/10		29/10		31/10	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99	Submit an *application for finance* for a customer	srp	1/10	1/10	2/10	2/10	4/10	4/10	8/10	8/10	9/10	10/10	9/10	10/10
100	Approve an *application for finance* for a customer	srp	1/10	1/10	2/10		4/10		8/10		9/10		9/10	
		Blocked: Waiting for management decision on exact routing of application approval notification												
101	Accept *financing* for a customer	srp	1/10	1/10	2/10	2/10	4/10	4/10	8/10		9/10		9/10	
		Behind Schedule: Barry on sick leave for 4 days												
		Behind Schedule: Barry still catching up after being ill last week												

Gary's Garage Feature List - Progress (Date 26/10)

We can also ask the Chief Programmers to supply the release manager with a one-line reason to explain why the feature is behind schedule at each release meeting. We print the reason underneath the feature on the wall chart. For every release meeting that the feature remains behind schedule, another one-line reason is added. The result is that the red area on the wall chart grows over time until the problem is sorted out. The growing patch of red shouts louder and louder at the Chief Programmer and the Project Manager until someone does something about it. Once the feature is back on schedule, the reasons are no longer printed.

Feature Kills

Visibility of results is a great motivator to encourage continued or increased levels of performance. One way this can be approached is through *feature kills*, which can be a fun and effective way of promoting team interaction, healthy competition, and better performance.

After each release meeting, the release manager prints out a sheet of paper containing a green icon for every completed feature that a developer has worked on. A second row of red icons represents every feature the developer is currently working on that is behind schedule. Each developer must display all "kills" and "misses" prominently at his or her workstation.

As well as being a bit of fun, this sends a couple of messages to the developers:

1. It is important to meet planned dates.
2. The performance of the feature team dictates the number of kills and misses you get, so it is the performance of the feature team that is important; not the performance of anyone individual on the team.

The feature kills method is not something we need to keep running for the life of a project. Once the message has gotten through and hitting deadlines has become the norm rather than the exception, we can stop playing the feature kills game. The novelty does wear off after a while, anyway, and there is no point spending time to produce the "kill sheets" every week if the team is no longer benefiting from it.

Summary

FDD uses six sharply defined milestones to track progress of each feature through processes 4 and 5, Design by Feature and Build by Feature.

By assigning percentages of completeness to each of the milestones, we can tell how complete a feature is. Rolling these percentages up allows us to show how complete each feature set is, how complete each feature area is, and how complete the entire project is.

We can use this information to provide easily understood progress reports for various levels of management within and external to the project

Collecting the data at regular points over time allows us to plot various graphs and to measure the rate of feature completion over time.

FDD does not mandate the style of reporting, but it certainly enables highly accurate progress tracking and reporting, using a few simple tools and a minimum of administrative overhead