Manage It!

Your Guide to Modern, Pragmatic Project Management

Manage It!

Your Guide to Modern, Pragmatic Project Management

Johanna Rothman

The Pragmatic Bookshelf
Raleigh, North Carolina   Dallas, Texas

Chapter 11

Creating and Using a Project Dashboard

Most of the questions you answer come down to some variant of this question: “Where are we?”

This could come from senior management trying to figure out whether you’re going to hit a deadline or from the project team asking about their status. It soon feels like they’re in the backseat of your car on a long trip asking, “Are we there yet?”

The key to understanding a project is to make regular measurements—both quantitative and qualitative—and display the measurements publicly. When project managers display these measurements as part of the project status, teams are able to adjust their work and proceed more successfully.

This collection of measurements comprises your *project dashboard*. Taken together, the project measurements display your velocity, distance, consumption, and location—much as a car dashboard does.

Creating a project dashboard provides feedback to the team and reports status to other interested people. Use a Big Visible Chart or Information Radiator [Coc04] so that everyone can see the project’s progress.

11.1 Measurements Can Be Dangerous

Measurement involves three big problems: the project team spending too much time on measurement to the detriment of the work, gaming the system, and measuring the people instead of the project.

It's easy to spot and fix people spending too much time measuring. Are your project staff members generating paperwork and measurements rather than performing the work? The measurements in this chapter are all obtainable by you, the project manager. Most of the project staff shouldn't have to help you obtain project measurements. You might need help from someone who manages the SCM or the DTS. (If you need to measure performance or reliability, you might need developers or testers to help you measure that.) If you need help from more than a couple of people, work on your project infrastructure support. The goal of the dashboard is to use the data to assess the project state, not to spend time creating the dashboard.

It is often harder to spot people gaming the system, but the cause is often that only a single factor is being measured. You're likely to see schedule games (see Chapter 6, *Recognizing and Avoiding Schedule Games*, on page 103) and other behavior that doesn't help the project progress. There's a famous Dilbert strip where the boss says he'll pay each developer some amount of money to fix defects. Wally, one of the characters says, "I'm going to write me a minivan." Wally is planning to write a whole lot of defects and then fix them. If you measure only one thing, you encourage people to optimize for that one thing. Make sure you have multiple measurements for assessing project progress.

When you choose measurements, make sure you measure the project and the product, not the people [Aus96]. If you measure anything that can be traced directly back to one person or another, you are measuring people, not the project or the product. Measuring people begs them to game the system, preventing you from understanding the project's state and possibly preventing the project from completing. Never measure people.

It's easy to measure some facets of a project, such as the project start date, the current date, and the desired release date and say, "We're X percent of the way along," because the project team has used that percentage of time. (See Section 11.2, *Earned Value for Software Projects Makes Little Sense*, on page 220.) If all you measure is the schedule, you're guaranteed not to meet the desired deadline.

In fact, measuring any single dimension can't give you a full enough picture of the status of your project. If you're driving a car and look at the mileage for this tank of gas but don't look at your miles per gallon and the miles left to drive, you still don't know whether you have enough gas to get you to your destination.

To obtain a true picture of the your project's state, choose at least four out of six dimensions of the project drivers, constraints, and floats from Section 1.2, *Manage Your Drivers, Constraints, and Floats*, on page 21 to measure and display on your project dashboard. Those four dimensions capture the areas you are most likely to be able to modify during the project. And if you don't measure them, you can't see what to change to make your project succeed.

Tip: Use Multidimensional Measurements to Assess Project Progress

> There are any number of references that say, "You get what you measure." And, as you saw in Section 11.1, *Measurements Can Be Dangerous*, on page 214, it's possible people will want to game the measurement. Since you will get exactly what you measure, make sure you measure enough information about the project to provide an honest assessment of project progress.

Rob, a VP of engineering called me, confounded. "JR, those freaking testers! They can't do anything right." Rob's project had 1,500 developers and about 350 testers. I had met a few testers before, so I said, "That's funny, the people I met seemed to know what they were doing." "No way," Rob quipped, "the developers meet every single milestone. The testers don't meet any. I need you right away to do an assessment."

Well, developers meeting every milestone is a suspicious statement. I know a lot of developers. And even the best don't always meet their schedules. I started the assessment with an open mind. Maybe all 1,500 developers really are incredible.

But here's what I discovered. The developers have to report only dates to the project managers. That's it. And the project managers measure only dates from the developers and defects from the testers. There's no measurement of anything else on the project. When I talked to developers about their work, it all became clear. Danny grimaced and explained, "I have to start the feature when the Gantt says to; otherwise, I get marked down on my performance evaluation. I put stubs in, so I 'finish' it on time. When the testers report a problem, I fix the problem."

Joe Asks...

Can I Ever Start Measuring Over?

You can. I don't recommend it, because generally the system (the process and the people) that's creating problems—such as starting the project a month late—doesn't recognize the system is doing this. Without a chart to show why you've been behind since the beginning, you won't be able to change the system of how projects start.

Instead of remeasuring, draw a line on the chart with something like "Original Start Date" and "Actual Start Date." Then show the triggering event that led to the "Actual Start Date." (See Figure 11.7, on page 227.)

Rob's organization has broken projects (and products)—projects that don't deliver what Rob needs. As long as he persists in single-dimension measurement for a group of people (dates for developers and defects for testers), they will have broken projects. The only cure for Rob is to have the project managers measure all around the project so that they can tell more accurately where the project is.

11.2 Measure Progress Toward Project Completion

By using several measurements from the drivers, constraints, and floats, you can measure the team's progress toward project completion. Project completion is a function of how accurate your original estimate was and how much progress you've made. But measuring only the schedule progress is not good enough. The only accurate way to measure progress for a software project is to measure how many features the project team has completed, how good those features are, and how many features are left to implement.

Use Velocity Charts to Track Schedule Progress

If you're implementing by feature, a velocity chart (such as Figure 11.1, on the following page) is a great progress indicator to how much progress

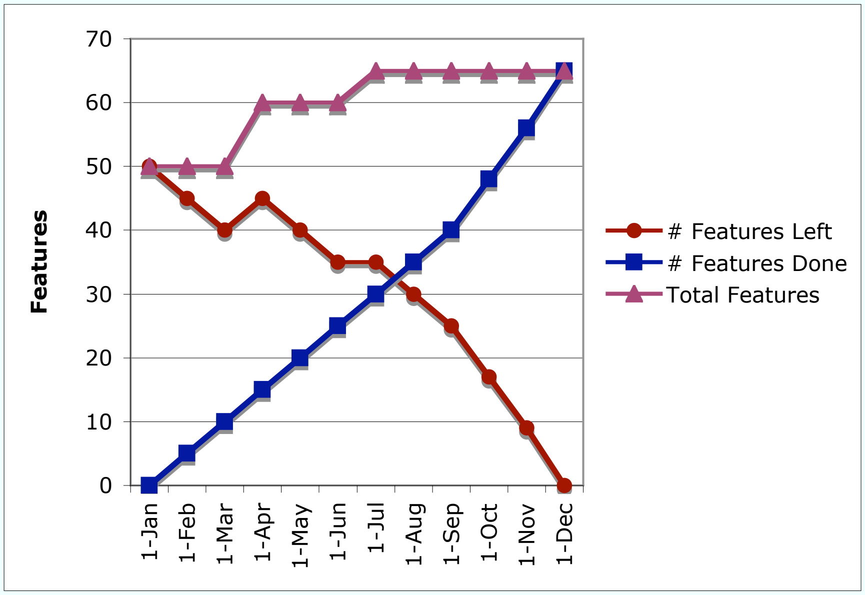

Figure 11.1: Velocity Chart for a Project

the team has made on the project.[1] And it can give you an indication about how much work is left.

Here's how you make a velocity chart. Add up the number of features—that's your total features. As you finish a feature, add 1 to the number of features done, and decrease the number of features left. If you have to add more features during the project, add those extra features to the total features. Even if your features aren't normalized to be the close to the same size, this chart will help.

If you use inch-pebbles and you're not implementing by feature, tracking inch-pebbles (Section 8.10, *Use Inch-Pebbles*, on page 173) can help you know where you are. But that won't be as accurate as implementing by feature. Whatever you do, don't just ask people whether they've met their milestones without looking to see how good the stuff is that they are producing.

1. See http://www.xprogramming.com/xpmag/jatRtsMetric.htm.

Tip: Velocity Charts Are the Single-Best Chart

If you can make only one chart, choose a velocity chart. Velocity charts use three measurements (requirements, completed work, and date), all on one chart. They don't provide a picture of defects or cost, two more measures you might like to see. But they provide an overall picture of progress on one chart.

Because you're measuring several trends on one chart: total requirements and *completed* work, including all the testing and documentation and whatever else your project requires over time, it's the single-best chart. If you're working without implementing by feature, the chart shows no completed work, which is exactly the state your project is in. Velocity charts are your friend.

Use an Iteration Contents Chart to Track Overall Progress

In addition to a velocity chart that tracks implemented features over time, you might want a finer-grained look at what's going on in each iteration. (Even if you're not using timeboxed iterations, generate this chart over a fixed time period. That will help you see when requirements changes and defects arrive in the project.)

In Figure 11.2, on the following page, you can see how the release's contents change over time. In this project, the team started with a velocity of six features per iteration. By they time they got to the ninth iteration, they were down to two features, plus two changes and four defects. At that point, the project manager realized things could only get worse and stopped changing the iteration's backlog during the iteration. That allowed the team to make much more progress in the last three iterations.

Until the project manager generated this chart, no one had any idea about the cost of the changes during an iteration and the introduction of defects those changes caused.

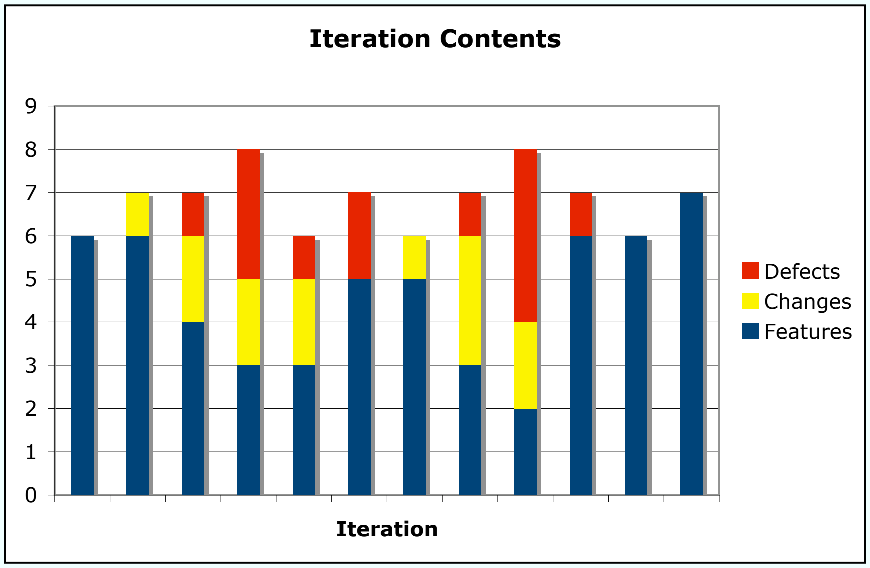

Figure 11.2: Iteration contents chart for a project

Earned Value for Software Projects Makes Little Sense

Earned value is a measure of the value of work performed to date.[2] But because software is ephemeral and ever-changing, it's close to impossible to calculate the true earned value. If you can't clearly define it, you can't really measure it. Resist the attempts from your organization to have you report on earned value. For a tangible product, it's easy to calculate earned value. If you're building a table, you can calculate the cost of the materials and time to see whether the legs and the top have value even before you put the table together. But earned value is different for software.

Here's an example. Say you have five requirements to complete in ten weeks. Imagine that you and the project team believe it will take the entire team two weeks per requirement. And, imagine you have five people on the team. Your estimate is ten effort-weeks per requirement, a total of fifty effort-weeks. Imagine the team has finished the first three requirements, including testing them, as in Figure 11.3, on page 222.

2. © 2007 R. Max Wideman, http://www.maxwideman.com; reproduced with permission.

Joe Asks. . .

How Can We Have No Completed Work?

You've been working hard for months on your project. No one has been slacking off. But when you try to use a velocity chart, it shows you no (or virtually no) completed work. How is that possible? It's possible—and even likely—if you're using a serial life cycle or implementing by architecture in any life cycle, without planning how to finish features.

When a project team uses a serial life cycle or implements by architecture, they have lots of partially completed work. Partially completed work is called *waste* in the lean community. It's waste because it's not done. Because velocity charts show you completed work, you can tell whether the team is producing waste or a completed product.

The more you use incremental, or even better, agile techniques, the more your velocity chart will show what you've done. Being able to show the project team what is done helps maintain the project rhythm and helps people accomplish more.

The customer sees what the team has done so far. "Looks great, but I really need it to do foo over here and blatz over there."

The "foo" and "blatz" features will cost another two team-weeks each. Your original calculation was that you had 60% of the work done in 60% of the time. You were on track. You are now not even close to on track. But you have feedback from the customer earlier than the end of the project, and you can give the customer what the customer wants.

How much value do you have? I don't know how to answer that question, because it doesn't account for the fact that the customer didn't realize what he or she wanted wasn't enough for the time allocated for the project. The initial measurements were wrong. Your project has some value. Maybe the work to date has even more value than you thought because the customer realized early that the requirements weren't quite right. But you are no longer 60% done; you're at some other percentage.

Some organizations like to use "Percent Complete." I don't agree with that either. All too often this refers to only the development piece, but

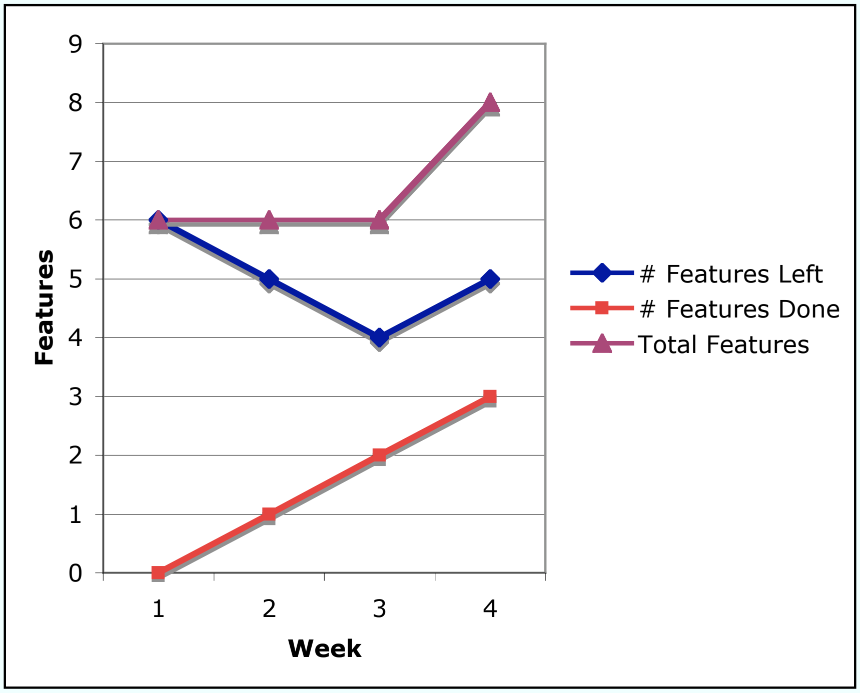

Figure 11.3: A six-week velocity chart

not the testing part. Pieces of the product that haven't been tested aren't complete. Using "Percent Complete" begs people to start with schedule games such as the one covered in Section 6.14, *90% Done*, on page 131.

If you want to know your progress, use a velocity chart showing running tested features. A velocity chart shows the team's actual progress against the planned progress. And it shows that change happens to a project and how much change is occurring.

So, just say no to earned value. Use velocity charts instead.

Track Your Original Estimate with EQF

Tom DeMarco in [DeM86] described a measure called *estimation quality factor* (EQF). EQF helps you understand how good the initial estimate was. At periodic intervals during the project, the project team answers

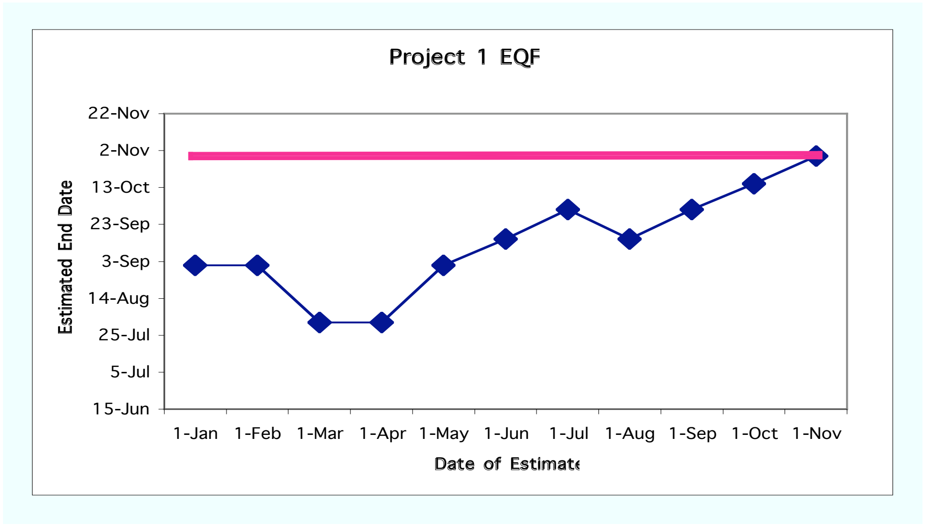

Figure 11.4: Estimation quality factor

this question: “When do you think we’ll be done?” Each data point is the consensus agreement on when the project team believes the project will be finished. At the end of the project, draw a line backward from the release date to the beginning of the project. For an example, see Figure 11.4. The area between the line you drew and the when-will-we-be-finished line is how far off your estimation was. This is a great technique for people to use as feedback on their individual estimates. But even if you don’t use it for feedback, it’s a great technique for the project manager to see what’s happening.

Maybe you’re concerned: there’s a penalty in EQF for discovering new requirements later. That’s true. EQF is not a perfect measure. But if you’re not going to use an agile life cycle, late requirements (or late-learned requirements) do bring a penalty. I’d rather see why the project is suffering from a delay than not know why.

If you’re using an agile life cycle, your velocity charts will provide you a quantitative answer, rather than a qualitative answer. But if you’re not using an agile life cycle, EQF is a great qualitative measure of how close your estimate is.

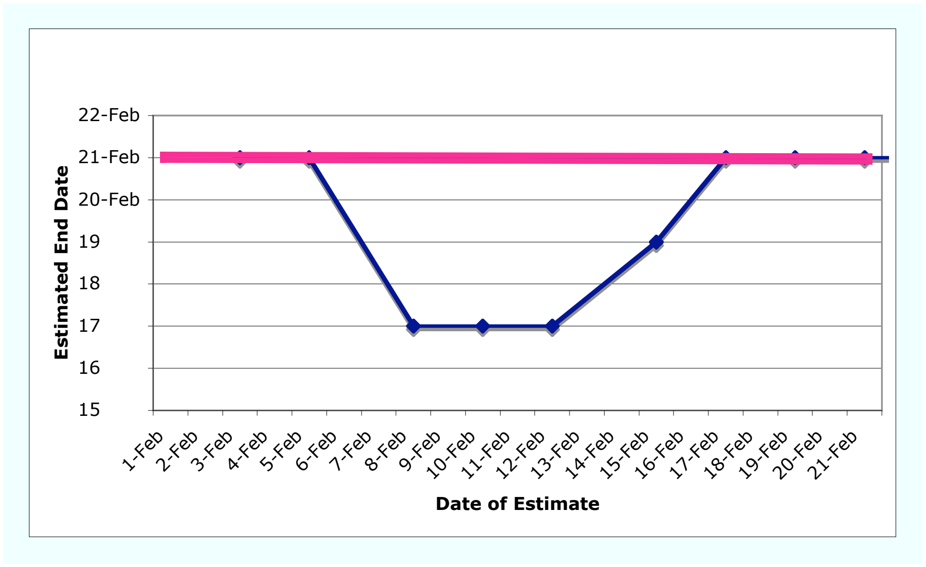

Figure 11.5: Tommy's estimation quality factor

Figure 11.4, on the preceding page is a chart of an EQF for a project that was originally supposed to be nine months long. For the first couple of months, when the project manager asked when people thought they'd finish, they said September 1. And for a couple of months, they were optimistic, thinking that they might finish early. But during the fifth month, team members realized they didn't know enough about some of the requirements. What they discovered changed the architecture and pushed out the date. For the next few months, they still weren't sure of the date. They realized in the last three months of the project that, because of the changing architecture, they were encountering many defects they hadn't anticipated. Evaluating EQF, a qualitative metric, was helpful to the project manager and the project team as a check against the progress charts.

EQF isn't just for software projects. A person can use this technique when performing any project work. I used it when writing this book. You can use it with developers (or testers or writers or whomever) to coach them about their estimation.

Tommy was working on a feature that he thought would take him three weeks to complete. He made sure he had several deliverables each week, his inch-pebbles. As he completed an inch-pebble, he updated his EQF for the feature. See Figure 11.5, on the previous page. He thought he was lucky with delivering the pieces early. He didn't change his EQF until about halfway through the feature, even though he had managed to complete most of his deliverables early for the first part of the feature.

As Tommy proceeded, he didn't quite make the progress he thought he would. He was still on track for his original estimate but was not going to meet the earlier date.

Schedule estimates are just guesses, so anything you can do to show and then explain why your schedule varies from the initial plan will be helpful to anyone who wants to know "Where are we?"

More Measurements Tell the Rest of the Story

Project completion measurements might be all your managers want to see, but if you're a project manager or a technical lead on a project team, I'm sure you'd like some early warning signs that the schedule might not be accurate. To keep my finger on the pulse of a project, I monitor several measurements:

- Schedule estimates and actuals, aside from EQF. If you use velocity charts, you get this as part of velocity.
- When people (with the appropriate capabilities) are assigned to the project vs. when they are needed.
- Requirements changes throughout the project. If you use velocity charts, you get this as part of velocity.
- Fault feedback ratio throughout the project if you're not using an agile life cycle. See Section 11.2, *See Whether the Developers Are Making Progress or Spinning Their Wheels*, on page 230.
- Cost to fix a defect throughout the project, especially if you're not using an agile life cycle.
- Defect find/close/remaining open rates throughout the project.

Note that these are assessment measurements, not measurements that are trying to find the problems in the project. These measurements will expose problems but might not be sufficient by themselves to see the real problems. The power from the measurement comes from looking at all of these measurements together.