



Using Cumulative Flow Diagrams

By David J. Anderson, May 2004

Introduction

Agile software development methods such as Scrum and Feature Driven Development manage and report project progress in a very different manner than traditional critical path project management. Scrum started with the use of a “burn down” chart which plotted the estimated number of hours remaining to complete the Sprint backlog. More recently “burn up” charts have become popular. These plot the number of completed Stories, Tasks or Features on the project with a projected completion target. It is possible to extrapolate the plot with a trend line to estimate the completion date of a project.

The concept of a “burn up” chart had been used in Feature Driven Development since its inception in the late 1990s. It’s called a Feature Complete Graph as shown in Figure 1.

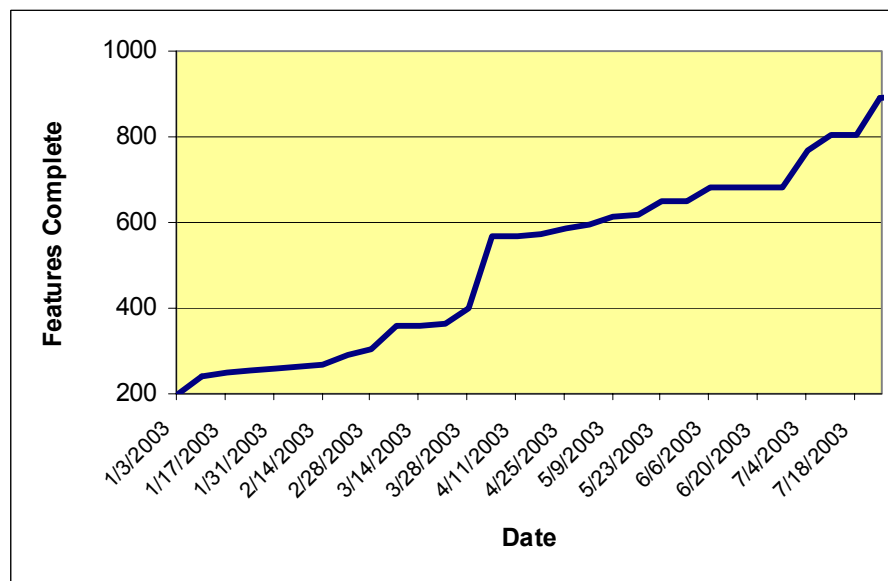


Figure 1. FDD Feature Complete Graph

However, “burn up” charts aren’t really sufficient for managing a project. They have no concept of work-in-progress (WIP) and simulating the anticipated end date is problematic. With Agile Management [Anderson 2003] I introduced Cumulative Flow Diagrams as a better replacement. This article explains why.

The S-Curve

The completion of Features tends to follow an S-Curve model, as shown in Figure 2. The S-curve effect makes it difficult to predict the end-date of a project from a single plot of Features complete.

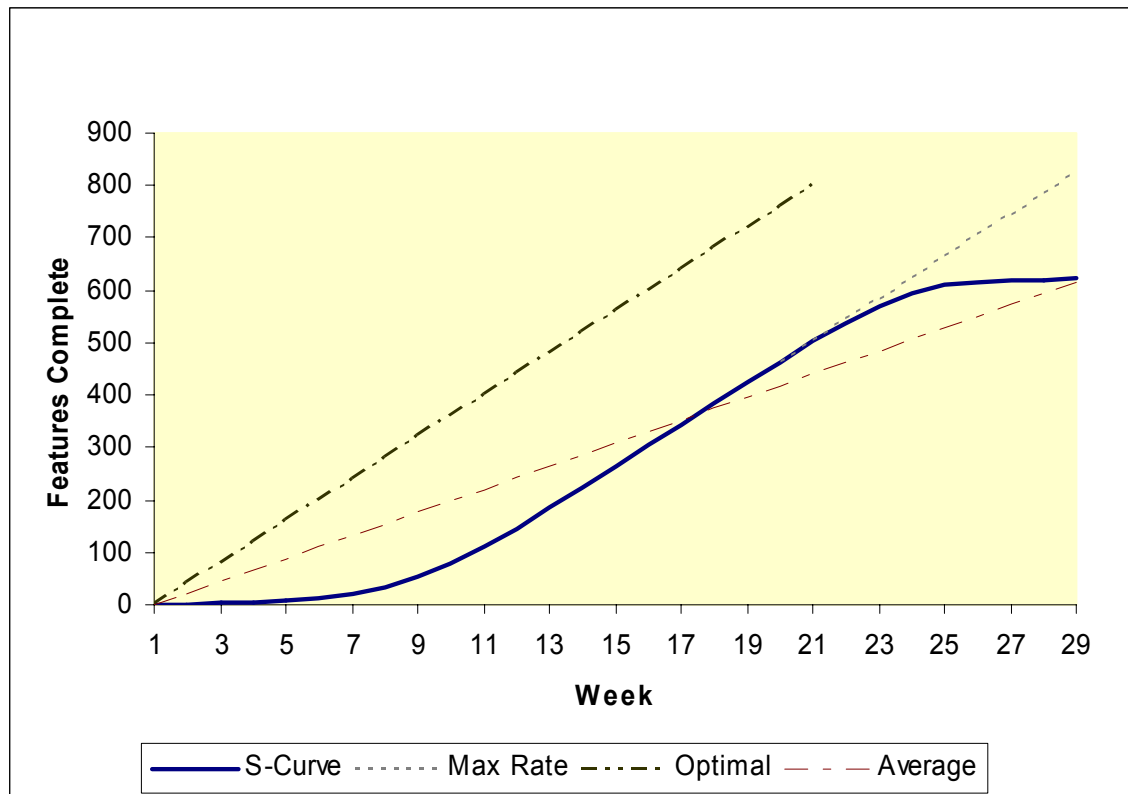


Figure 2. S-Curve for Features Complete

In order to communicate, a fuller picture of a project’s health, I found it necessary to supplement a Feature Complete Graph with a report of the percentage complete and an accompanying graph. A percentage complete plot for FDD gives credit to Features which are in-progress. Each of the six milestones for a Feature is credited with an earned-value percentage, as shown in Figure 3.

Domain Walkthrough	Design	Design Inspection	Coding	Code Inspection	Promote To Build
1%	40%	3%	45%	10%	1%

Figure 3. Feature Milestones and earned-value percentages

So Feature Complete graphs fail to communicate work-in-progress but the Percentage Complete figure and graph suffer from “false reporting”. Why? Because any agile developer will tell you that they only value finished working software. So there is no value in partially complete code. This result is compatible with the Theory of Constraints management accounting method, Throughput Accounting [Corbett 1997]. Value should only be recognized on delivery.

Hence, it was necessary to stop reporting percentage complete altogether. Only report the true earned value – Features complete – or find a better way to communicate the work-in-progress, i.e. a method which did not report the value of WIP but did communicate that work was progressing even when Features were not being completed every day.

Work-in-progress

If it is wrong to communicate earned value from partially complete WIP, then we should ignore it? No, I don't believe so! Rather, we should care about it deeply. Why?

Work-in-progress can be thought of as inventory. In this case, it is knowledge work inventory – ideas for valuable working software, captured at some stage in a transformational process which takes it through transformations such as analysis, design, test plan, code, unit test and code inspection (depending on the method you are using). These ideas are developed from the work of Donald Reinertsen but to understand that, we must first understand the contribution of Marvin Patterson.

Patterson's Design Model

With *Accelerating Innovation* [1993] Marvin Patterson introduced a concept for modeling the design process. He asked us to envisage that design was a process of information discovery. Before a design is started there is little or no information, perhaps only a vague thought. As the design emerges there is gradually more and more information and less and less uncertainty until the design is complete.

Mary Poppendieck [2003] suggested that software development can be understood with the same model. In other words, all software development is a “design problem”. This would allow us to model the flow of value through a software engineering system as the gradual reduction of uncertainty and the discovery of more and more detailed information until working code which passes appropriate quality control tests, is produced.

Design is Perishable

Writing in *Managing the Design Factory* [1997] Donald Reinertsen developed the ideas of Patterson a little further by introducing two concepts. The first observation was that design-in-process inventory could be tracked using Cumulative Flow Diagrams, as shown in Figure 4. CFDs were already in use in Lean Production to track the flow of value through a factory. The second and perhaps the most valuable insight was that the value of design information depreciates over time. There are several reasons for this. The main one is that information need only be created once, as the cost of replicating it is near zero. If design is information then the time it takes a competitor to duplicate the design, is the

time in which the design (and its associated information) has a differentiating value. Design information is also perishable because of possible changes in the marketplace – fashions change and so do laws, regulations, materials, supply components, distribution networks and business models. To have real value a design must be appropriate for its time and it must come to market within its window of appropriateness. If software is design then the same must be true of it. The requirements for a software program must be perishable. Hence, the faster the requirements can be realized as working code and brought to market, the more value will be delivered. Reinertsen's and Poppendieck's observations tie software engineering firmly to the principles of Lean and the lead time for turning an idea into working software must be a critical to the financial success of any software activity.

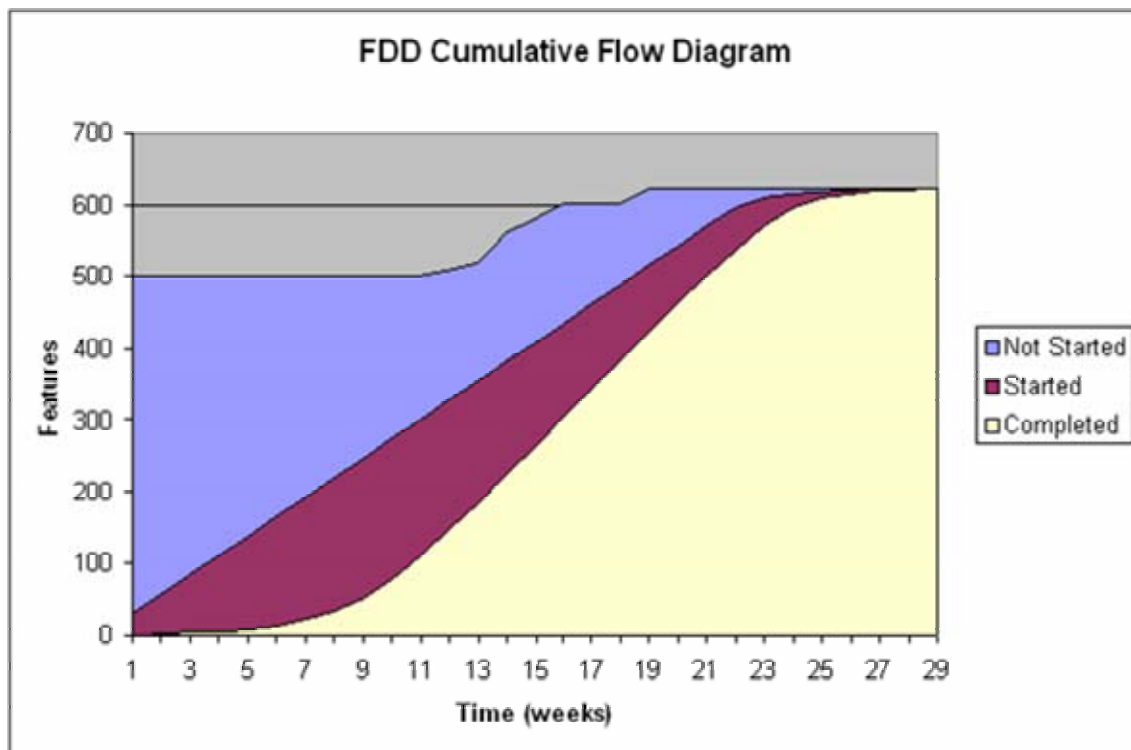


Figure 4. An idealized Cumulative Flow Diagram for an FDD project

Little's Law

Little's Law states that a queue of material can be analyzed in two ways: as inventory; or lead time. Simply put the size of the inventory is directly proportional to the lead time for processing that inventory. Hence, the size of work-in-process inventory matters because in agile development we want to complete software in short cycles and deliver value as often as possible. Figure 5 shows how to read the WIP inventory and Lead Times from a CFD.

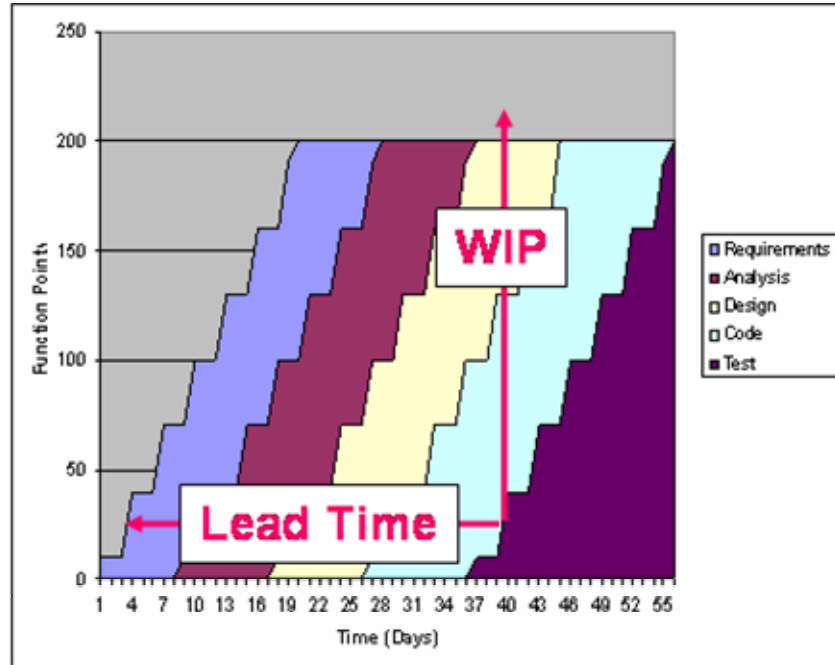


Figure 5. Reading WIP and Lead Time from a CFD for Day 40 of a project

Batch Size

Figure 5 also demonstrates how batch size and batch transfers affect the cumulative flow plot. The batch transfers can be clearly seen from the jaggedness of the plot. With larger batch sizes, there is more WIP and longer lead times. With smaller batch sizes as in Figure 6, WIP is reduced and lead time falls accordingly. Note the smoothness of the plot in Figure 6. This is from a real FDD project with Chief Programmer Work Packages (small batches of Features) which were completed in less than 2 weeks. The lead times can be clearly read from the diagram.

It is easy to see from this that CFDs can be used to tell at a glance, the size of iterations and the type of method being used for development.

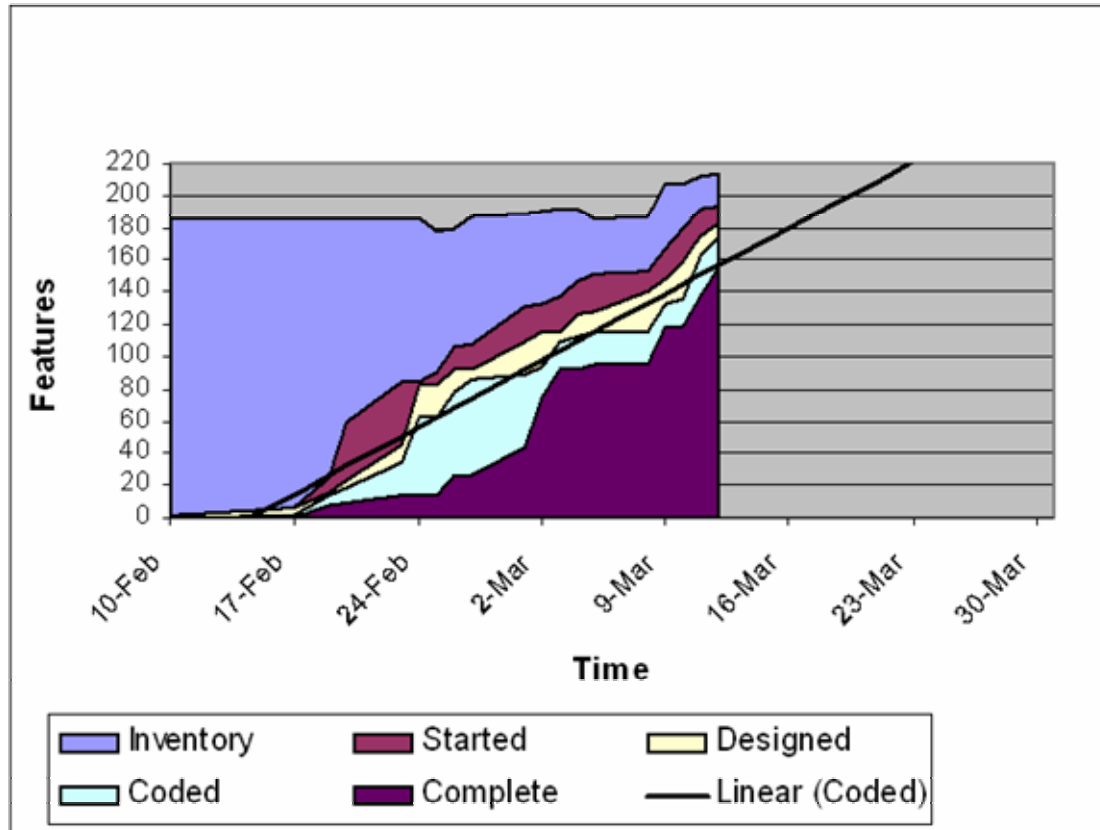


Figure 6. CFD showing lead time fall as a result of reduced WIP

WIP is a Leading Measure

Reinertsen describes WIP as a leading metric. What he means is that WIP can predict in advance the lead time and delivery date. It can therefore be used to correct problems before they become too serious. If we waited to measure lead time or delivery date, there may be greater problems stored up. I once had a project which was reporting 53% complete but only 4 Features were completed. A CFD plot of this project would have alerted me, as the manager much earlier. I will discuss how to monitor the health of projects using CFDs in a future Coad Letter.

Summary

Cumulative Flow Diagrams provide a method for tracking progress of agile projects in a “burn up” fashion. Because they plot both the total scope and the progress of individual Features / Stories / Tasks / Functions / Use Cases they communicate absolute progress whilst visually providing a proportional message of total completeness. CFDs also offer us a simple method of tracking work-in-progress and visually analyzing the trend in lead time for delivery of working code. They provide a leading metric which allows teams and managers to react early to growing problems and provide transparency into the whole lifecycle. Tracking a project with a CFD is a key element in moving to a Lean system for software development.

About the author

David J. Anderson is the author of the recent book, “Agile Management for Software Engineering – Applying the Theory of Constraints for Business Results” published in Peter Coad’s series by Prentice Hall PTR in September 2003. He is Principal Consultant with VA Systems Professional Services. David was one of the team which created the popular agile development method, Feature Driven Development. He has introduced FDD at two Fortune 100 companies Sprint (a telecommunications operator in the United States) and Motorola. He writes the regular *Agile Management* column at the Borland Developer Network website and publishes the his weblog at <http://www.agilemanagement.net/>.

He holds a degree in Computer Science and Electronics from the University of Strathclyde.

Email: dja@agilemanagement.net

References

- [Anderson 2003] Anderson, David J., *Agile Management for Software Engineering – Applying the Theory of Constraints for Business Results*, Prentice Hall, Upper Saddle River, NJ, 2003
- [Corbett 1997] Corbett, Thomas, *Throughput Accounting*, North River Press, Great Barrington, MA 1997
- [Patterson 1993] Patterson, Marvin, *Accelerating Innovation*, Van Nostrand Reinhold, New York, NY 1993
- [Poppendieck 2003] Poppendieck, Mary and Tom Poppendieck, *Lean Software Development – an agile toolkit*, Addison Wesley, New York, NY 2003
- [Reinertsen 1997] Reinertsen, Donald G., *Managing the Design Factory – A Product Developer’s Toolkit*, Free Press, New York, NY 1997