Feature-Driven Development:

towards a TOC, Lean and Six Sigma solution for software engineering

By David J. Anderson, Microsoft Corporation, October 2004

Abstract

Too often TOC practitioners assume that there is no TOC application for software engineering and jump immediately to the Thinking Processes to find answers for problems in software development. Others believe that the only way TOC can help is generic project management and offer Critical Chain as the solution. “Agile Management for Software Engineering” [Anderson 2003], showed that software development can be modeled as a flow problem and that to do so requires knowledge of how software is constructed – the architecture of software systems. Once modeled as a flow problem, the Drum-Buffer-Rope solution can be applied with great success. Recent developments of so called “agile” methods in software engineering are compatible with TOC teaching and often implement a rudimentary DBR solution without understanding it, from a TOC perspective.
Cumulative flow diagrams (CFD’s) from Lean Production can be used to report the flow of value through a DBR software development solution. CFD’s can be used to provide insight into constraints which are otherwise hard to see due to the nature of knowledge work. The data from the CFD’s has been used to develop an understanding of variation in software engineering. The use of Shewhart’s statistical process control, Deming’s Theory of Profound Knowledge and Wheeler’s four states of control are introduced for the assessment of capabilities and management of reduction of variation in the process of software development.
Analysis techniques used to model the requirements and architecture are used to measure the capacity of the system and identify the constraint. Once this is done, Throughput Accounting can be used for product mix selection and Critical Chain introduced for multi-project scheduling selecting a shared resource such as user interface design as the synchronizer for the staggered schedule.
[This paper assumes familiarity with TOC Fundamentals and does not explain terms such as “the five focusing steps.” However, it does not assume any knowledge of software engineering.]

An Introduction to Feature Driven Development

In 1997 Jeff De Luca, an Australian, put together a team for a large lending system project at United Overseas Bank in Singapore. The team included Peter Coad who was well known for his writing on object-oriented analysis and design and the development of a process of software engineering for use with object technology on large line of business IT projects. He called this process method, The Coad Method. At the heart of The Coad Method was an analysis artifact he dubbed "a feature". Features sounded like requirements and were written using domain language which the project sponsors could understand. The concept with features is that each one was written so that the sponsor could agree that the feature had meaning and was required in the system. During *the Singapore Project* between 1997 and 1999, The Coad Method evolved into Feature Driven Development with the introduction of ideas from the work of Gerald Weinberg, Frederick Brooks, Timothy Lister and Tom De Marco with some valuable insights on reporting from Jeff De Luca and batching of work from Stephen Palmer. Peter Coad and Jeff De Luca later published a brief outline of FDD in their 1999 book "Java Modeling in Color with UML" [Coad 1999]. Stephen Palmer with Mac Felsing wrote the definitive textbook, "A Practical Guide to Feature Driven Development" [Palmer 2001] two years later.

The Agile Software Development Movement

In the winter of 2001, a group of leading thinkers in the software engineering community had come to realize that many of them had been talking about similar things and that these things were paradigm shifting in comparison to the direction the industry had taken in the 1990's. They decided to meet for a summit at the Snowbird ski resort in Utah. The result was a declaration known as the Manifesto for Agile Software Development. One of the signatories, Jon Kern, worked for Peter Coad at Togethersoft. As a result, FDD had become one of the recognized set of agile software methods, along with others such as Scrum, Extreme Programming and Adaptive Software Development.

What is *Agile* Development?

At the heart of agile software development are two core ideas – delivering value in the form of working software is most important – and responding to change in a fickle market full of innovation and economic uncertainty (the Internet and telecom bubbles had been in full swing for several years previously) must be built in to any method for software development. This meant breaking the traditional notion of "plan the work, and work the plan" and moving to an iterative delivery approach with adaptive planning. The core enabler for many agile methods was a definition of value recognizable to the sponsor or customer. Extreme Programming has "user stories", whilst FDD has "features".

The Feature as Process Enabler

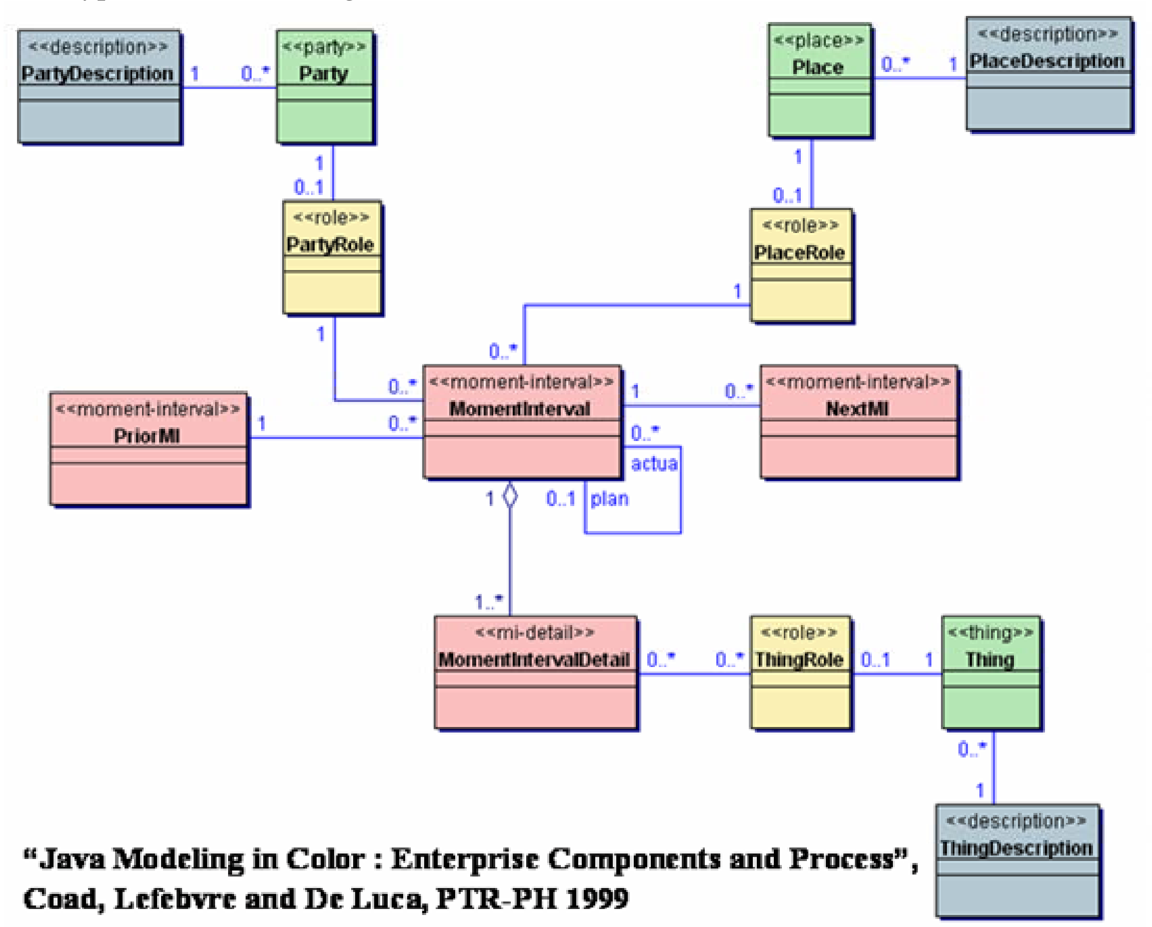
Features are tiny! In The Coad Method, the definition of a feature is written using this template:

<action> [a|the] <result> [of|to|for|from|…] <object> [with|for|of|…] <parameters>

e.g., list the available conference venues for a given hotel chain (for a given set of dates and anticipated attendee numbers)

Features map directly on to an object domain model for the system. Features are written using the language of the domain and are unambiguous in terms of their expression of the code to be developed to fulfill the requirements. The <object> identifies the class on the domain model – in this case the HotelChain. The <action> identifies the method or function name on that class – in this case listAvailableVenues(). And the <result> describes the return value from the method – in this case ConferenceVenue[] (a collection of conference venue data or objects.

Features are defined based on a domain model. The domain model is created using a technique called the Unified Modeling Language (UML) class diagram and Peter Coad's enhanced technique called the Domain Neutral Component (DNC) and class archetypes, as shown in Figure 1.



"Java Modeling in Color : Enterprise Components and Process", Coad, Lefebvre and De Luca, PTR-PH 1999

Figure 1. The Domain Neutral Component

Figure 2 shows an example of the DNC in use for a hotel and conference domain example.

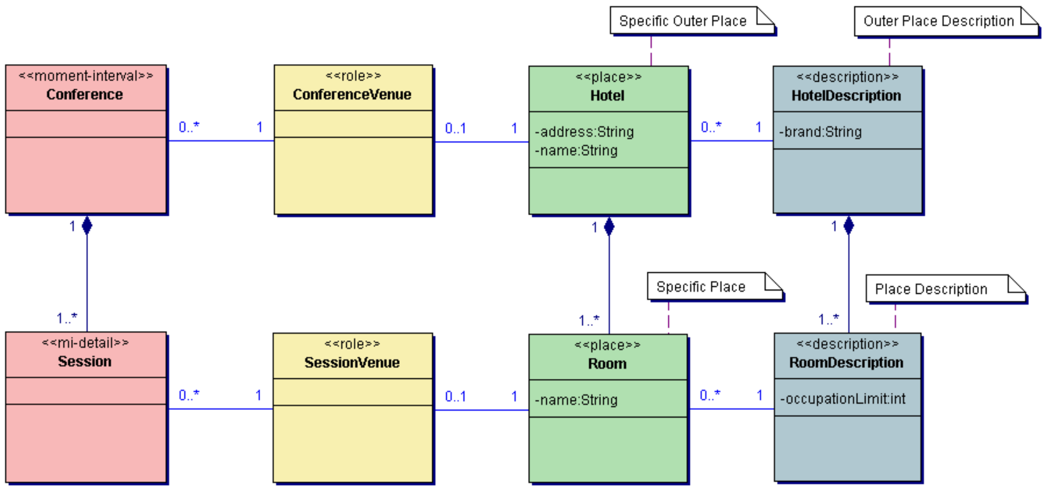


Figure 2. Hotel & Conference Venue DNC Model

Each feature in FDD reads as a requirement which is understandable by the sponsor – it has true business meaning and describes true business value. However, each feature also translates directly to a design artifact known as a UML sequence diagram. Figure 3 shows the sequence diagram for the feature described above, ***list availability of conference venues for a given hotel chain***.

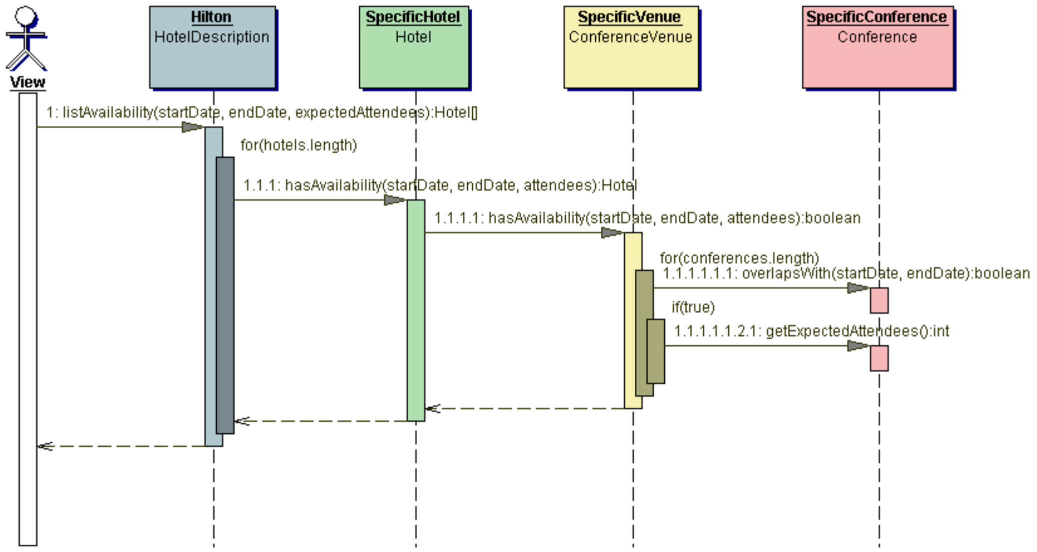

View
Hilton
HotelDescription
SpecificHotel
Hotel
SpecificVenue
ConferenceVenue
SpecificConference
Conference
1: listAvailability(startDate, endDate, expectedAttendees):Hotel[]
for(hotels.length)
1.1.1: hasAvailability(startDate, endDate, attendees):Hotel
1.1.1.1: hasAvailability(startDate, endDate, attendees):boolean
for(conferences.length)
1.1.1.1.1.1: overlapsWith(startDate, endDate):boolean
if(true)
1.1.1.1.1.2.1: getExpectedAttendees():int

Figure 3. UML Sequence Diagram for ***list availability of hotel conference venues***

Features represent small pieces of value in the system being delivered. Features and the use of a domain model, sequence diagrams and mapping of features into the UML significantly reduce the variation in development tasks. Features are dependable! They are predictable! The effort to build features can be estimated accurately with a low tolerance. Empirical data collected over the last 5 years has shown me that level of effort per feature varies between half a man day to eight days with the spread heavily biased to the bottom end and the mean feature taking around 1.2 man days.

Flow in Feature Driven Development

The FDD method defines 6 milestones for each feature:

1. walkthrough – explanation of the requirement to the developers (face-to-face)
2. design – creation of the sequence diagram
3. design review – peer review to check the design meets the requirements
4. coded – methods are written in class files to deliver the design
5. code review and unit test – test & peer review to check that code does what was specified in the design
6. promotion – into the integrated build for system / product testing

So FDD offers a mechanism for defining value in a fine-grained manner and for tracking the flow of the value through a set of transformative steps.
Features are grouped into collections known simply as feature sets. Each set of features is associated with a single <<Moment-Interval>> archetype class (pink) on the domain model. In turn features sets are grouped into collections known as subject areas. Each subject area is associated with a sequence of <<Moment-Interval>> archetype classes on the domain model. In this respect FDD is analogous to a V-Plant where very small components, called features, are constructed from a raw material, a feature description and subject matter expertise. They are then assembled into larger components of greater value called feature sets and then yet larger components of even greater value called subject areas, as shown in Figure 4. The flow of value in FDD happens in a V-Plant model.

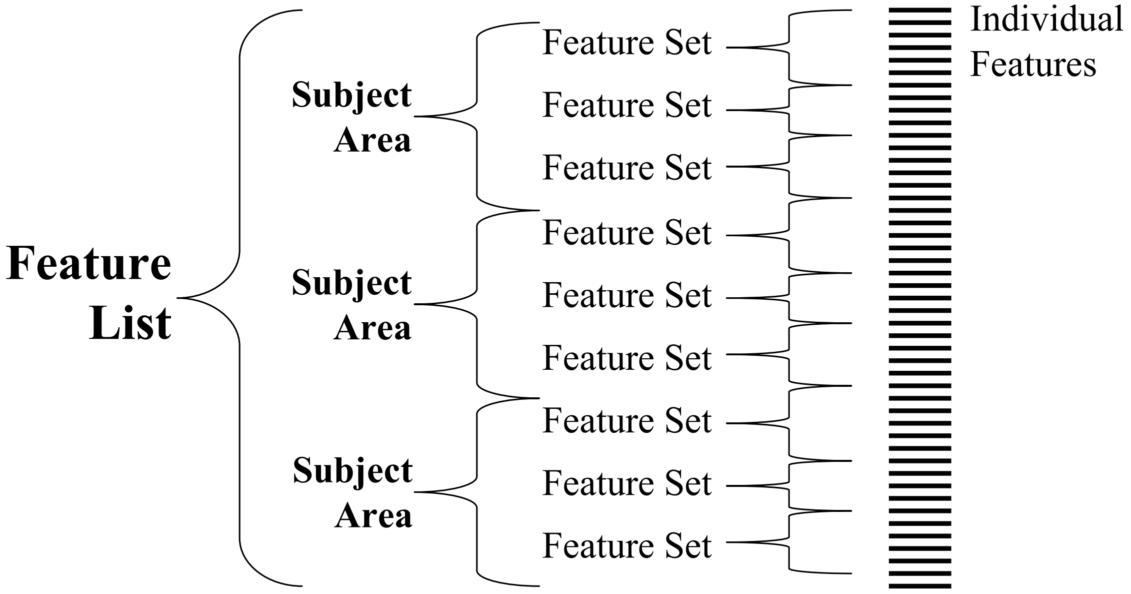


Figure 4. Component assembly in FDD

Once you have a flow model for a problem, it is possible to use a Drum-Buffer-Rope solution to improve the flow of value. The epiphany which linked TOC and FDD was that features must be treated as the "inventory" in the software development problem.

Tasks versus Features

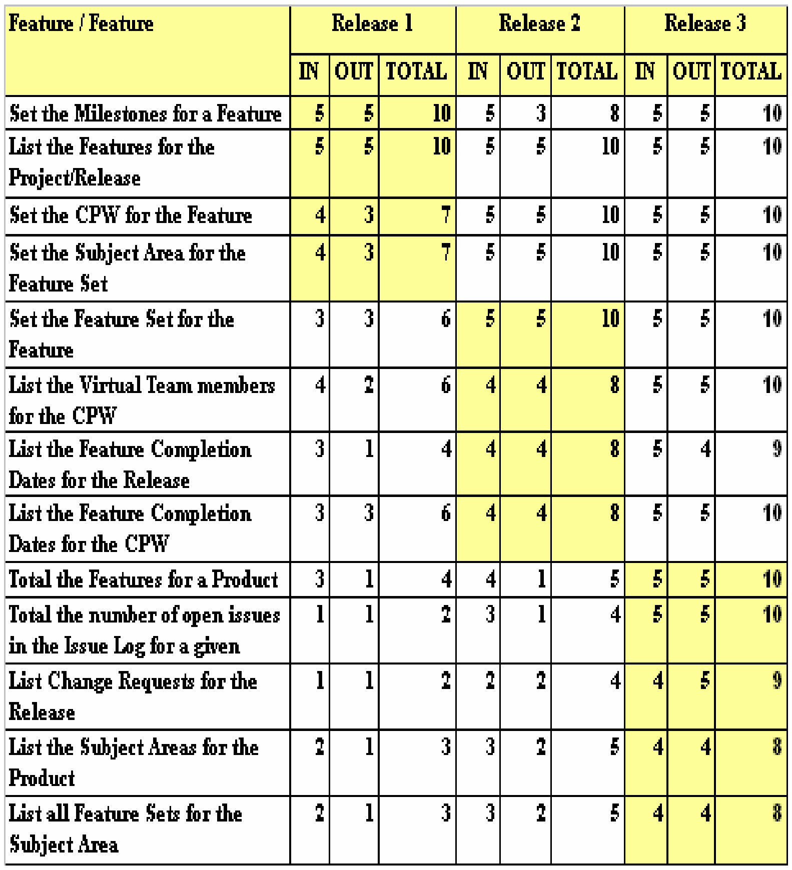
The traditional software engineering approach of breaking work down into an arbitrary set of developer tasks does not allow the use of the DBR solution because it is impossible to identify the constraint – there is no flow. It is possible to monitor task estimates versus actual execution. This allows the monitoring and possible reduction of variation in task estimates but it does not enable a solution to monitor the throughput and quantity of value delivered in a given time period. Features as an example of an analysis artifact that measure value are required to enable the DBR solution. As we shall see later, introduction of a DBR solution enables the multi-project solution for Critical Chain by providing a measure of capacity, loading and throughput for the synchronizing resource. Features enable a measure of capacity of an organization to process value items – though they do not identify the dollar value. [How to identify a prospective dollar value for features is discussed toward the end of this paper in the section on Throughput Accounting.]

Feature Product Mix using Kano Modeling

In the DBR solution once the capacity is identified, the flow at the input must be stemmed off to the limit of the constrained system to process it. In order to fully exploit the constraint the most valuable features must be processed through the system by priority. Peter Coad introduced the idea that features for a given product iteration or release should be selected using Kano modeling. Each feature is rated 0 to 5 based on how exciting is it that the feature be included in the release and 0 to 5 on how painful it would be if it were left out. Figure 5 shows a simple example divided in to 3 releases based on capacity.

FDD and the DBR Solution

Using a feature as the unit of inventory enables the measurement of the capacity of a software development department. Once this is done, the rate of input to the system can be reduced to match the capacity. Even though the constraint has not been identified inside the system of development, stemming off the input to the rate it can be consumed is a subordination activity and will help the system to cope with its constraint. The use of Kano modeling to prioritize input is an exploitation mechanism which helps to insure optimal utilization of the constraint - generating maximum throughput (in dollar terms) from the whole system. The amount of features input to the system must be less than the known maximum capacity. Features on the Kano model prioritized list which do not make the cut for the current work period, provide a buffer of extra inventory.
The objective from the first phase of the DBR implementation should be to stabilize the system before peering inside to identify the constraint and begin the process of continuous improvement using the five focusing steps.



Feature / Feature	Release 1			Release 2			Release 3		
	IN	OUT	TOTAL	IN	OUT	TOTAL	IN	OUT	TOTAL
Set the Milestones for a Feature	5	5	10	5	3	8	5	5	10
List the Features for the Project/Release	5	5	10	5	5	10	5	5	10
Set the CPW for the Feature	4	3	7	5	5	10	5	5	10
Set the Subject Area for the Feature Set	4	3	7	5	5	10	5	5	10
Set the Feature Set for the Feature	3	3	6	5	5	10	5	5	10
List the Virtual Team members for the CPW	4	2	6	4	4	8	5	5	10
List the Feature Completion Dates for the Release	3	1	4	4	4	8	5	4	9
List the Feature Completion Dates for the CPW	3	3	6	4	4	8	5	5	10
Total the Features for a Product	3	1	4	4	1	5	5	5	10
Total the number of open issues in the Issue Log for a given	1	1	2	3	1	4	5	5	10
List Change Requests for the Release	1	1	2	2	2	4	4	5	9
List the Subject Areas for the Product	2	1	3	3	2	5	4	4	8
List all Feature Sets for the Subject Area	2	1	3	3	2	5	4	4	8

Figure 5. Product Mix using Kano Modeling

The Patterson Design Model

With Accelerating Innovation [1992] 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 [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, that passes appropriate quality control tests, is produced. This approach is consistent with *cone of uncertainty* idea introduced earlier by Boehm [1981] and adapted by McConnell [1997].

Design is Perishable

Writing in his 1997 book, 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 used in Lean Production, as shown in Figure 6. 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.

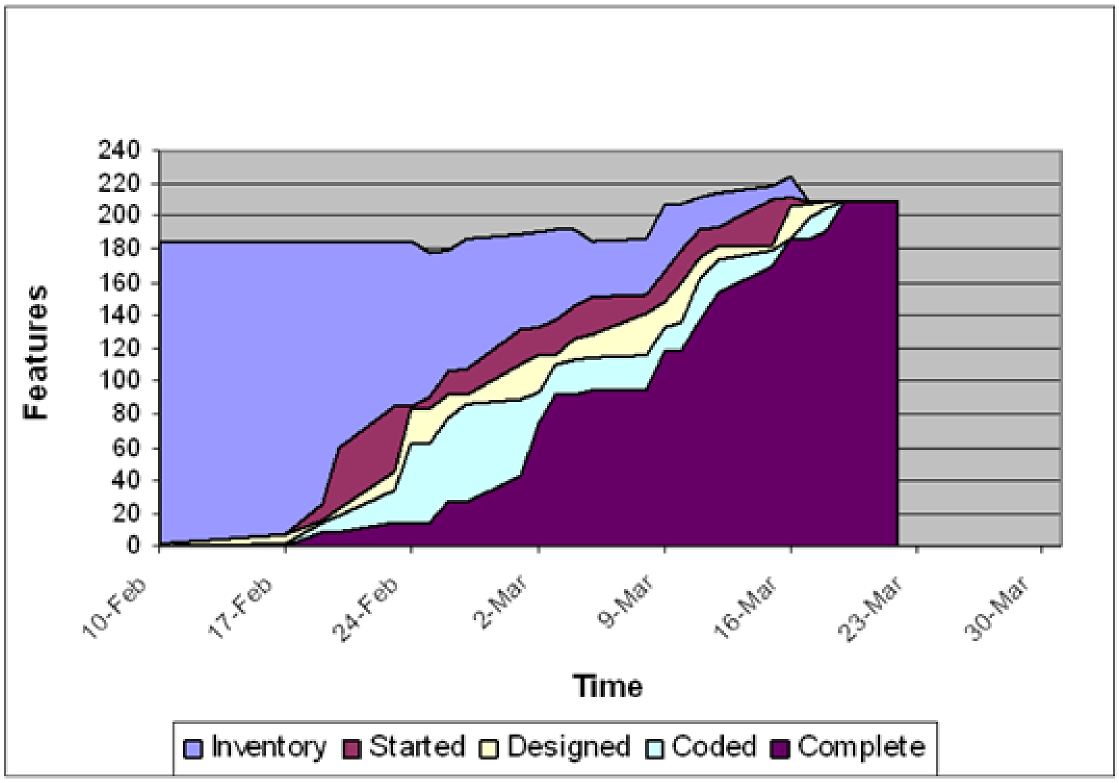


Figure 6. Cumulative Flow Diagram showing smooth flow

Batch Size & Quality

In software development, the organizational maturity to even track the flow of value is typically missing. Identifying the constraint within an organization can initially be impossible. Even identifying the capacity can be impossible. In the first instance, it makes sense to focus on reduced batch size and high quality assurance. Traditional software engineering methods use large batch sizes called "phases" e.g. analysis phase, design phase. This lifecycle model is usually referred to as "waterfall". FDD achieves small batch sizes and high quality through use of chief programmer work packages –

batches of features designed to be completed in two weeks or less by a single feature team – and a focus on quality assurance through unit testing and peer review. Other agile methods also focus on short cycle times and unit testing for quality assurance.
The effect of focusing on small batches and high quality is an immediate reduction in defects and increase in customer perceived quality through the reduction in the "perishable nature of requirements" affected by the reduced lead time of the small batches. Small batches create a "just-in-time" type solution for software development. By reducing defects, capacity is quickly freed up and stress is lifted off the whole organization. This creates a slack which opens a window of opportunity to affect more change and start to measure the flow of value through the system, to identify the capacity, and the capacity constrained resource – ultimately enabling the DBR solution.
Small batch sizes create a smooth cumulative flow, as shown in figure 6. When the batch sizes are large and the lead time long – in a waterfall fashion – the cumulative flow diagram is ragged as shown in figure 7. A project which displays a CFD similar to figure 7 is almost certainly delivering poor quality. Poor quality means high variability against the estimate and most likely indicates that the project promise is in jeopardy.

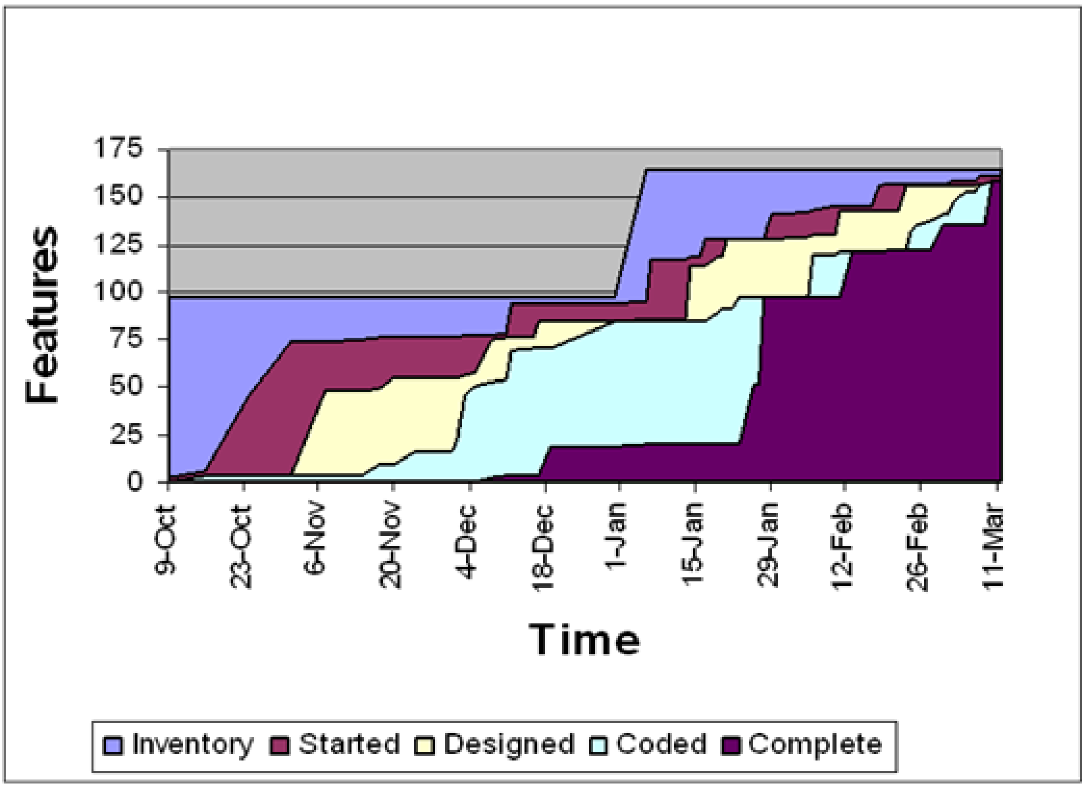


Figure 7. CFD showing ragged flow

Wheeler's 4 States of Control

Donald Wheeler [Wheeler 1992] has modeled what he calls the four states of statistical process control as shown in Figure 8.

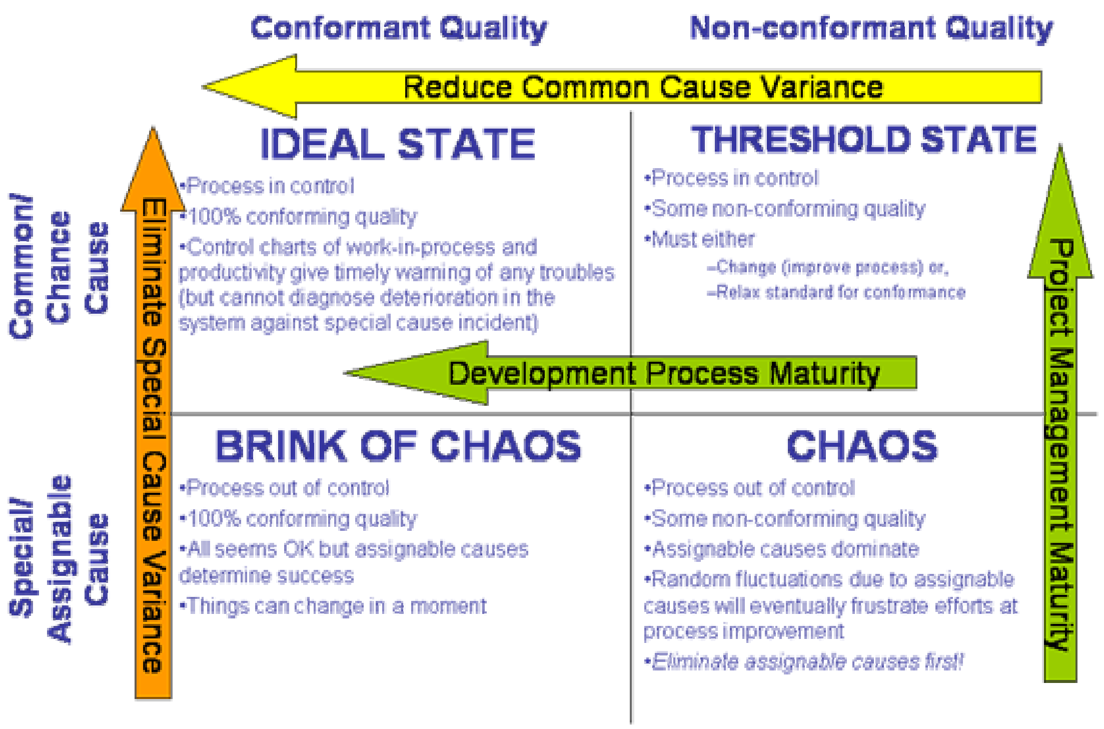

Conformant Quality
Non-conformant Quality
Reduce Common Cause Variance
IDEAL STATE
•Process in control
•100% conforming quality
•Control charts of work-in-process and productivity give timely warning of any troubles (but cannot diagnose deterioration in the system against special cause incident)
THRESHOLD STATE
•Process in control
•Some non-conforming quality
•Must either
–Change (improve process) or,
–Relax standard for conformance
Development Process Maturity
BRINK OF CHAOS
•Process out of control
•100% conforming quality
•All seems OK but assignable causes determine success
•Things can change in a moment
CHAOS
•Process out of control
•Some non-conforming quality
•Assignable causes dominate
•Random fluctuations due to assignable causes will eventually frustrate efforts at process improvement
•Eliminate assignable causes first!
Common/ Chance Cause
Special/ Assignable Cause
Eliminate Special Cause Variance
Project Management Maturity

Figure 8. Wheeler's 4 States of Control

Wheeler's model divides the world into two rows – common cause and special cause – and by two columns – conformant and non-conformant quality. The meaning of conformant quality is defined by the market within the particular industry. For software engineering we choose to define conformant quality as "on-time, on-budget, with agreed scope, and a quality level of x defects per hundred FDD features". To move from the right hand column to the left hand column, we must reduce common cause variation. To do this we can either, improve our system or lower our standards – redefine the meaning of conformant quality.

The notion of special cause variation creates a barrier to control. If special cause variation is allowed to affect a system it is never under control. Hence, to move from the lower row of the Wheeler chart, it is necessary to eliminate (not merely reduce) special cause variation. Edwards Deming referred to this as "stabilizing the system" [Deming 2000]. Special cause variation is important because the system is unstable while it persists. This can mean that the constraint can move without notice. A moving constraint cannot be managed.

Six Sigma

It is easy to see how the Wheeler chart helps to understand the underlying philosophy behind Six Sigma. Six Sigma is a method designed to create a culture of continuous improvement through a focus on reduction of variation. Instantly Wheeler's matrix shows that reliability can be improved by eliminating special cause variation. For project management this means issues must be identified early and resolved before they affect the critical path. Further improvement is then achieved through reduction of common cause variation. Both DMAIC and DMADV processes relate directly to moving a software development process from right to left on the Wheeler matrix. DMAIC can be used to measure variance in estimation against analysis artifacts such as Features in FDD (see later section) and DMADV can be used to insure code written delivers defined requirements. If a process suffers a special cause problem then the TOC Thinking Processes can be used to identify the root cause and a plan made to eliminate it or accept it as an *acceptable* risk, i.e. the investment required to eliminate the root cause is too great to justify against the potential impact on the system.

The Wheeler model also provides a tool for objectively assessing and monitoring the capabilities of different functions in a system of software engineering. Figure 9 shows different functions and their assessment in terms of statistical control. Functions which exhibit special cause variation need attention, as they make the whole system chaotic and may cause variation beyond the size of the buffer in a DBR solution or cause the constraint to move unexpectedly. Holes in the buffer are indicative of a special cause variation somewhere in the system.

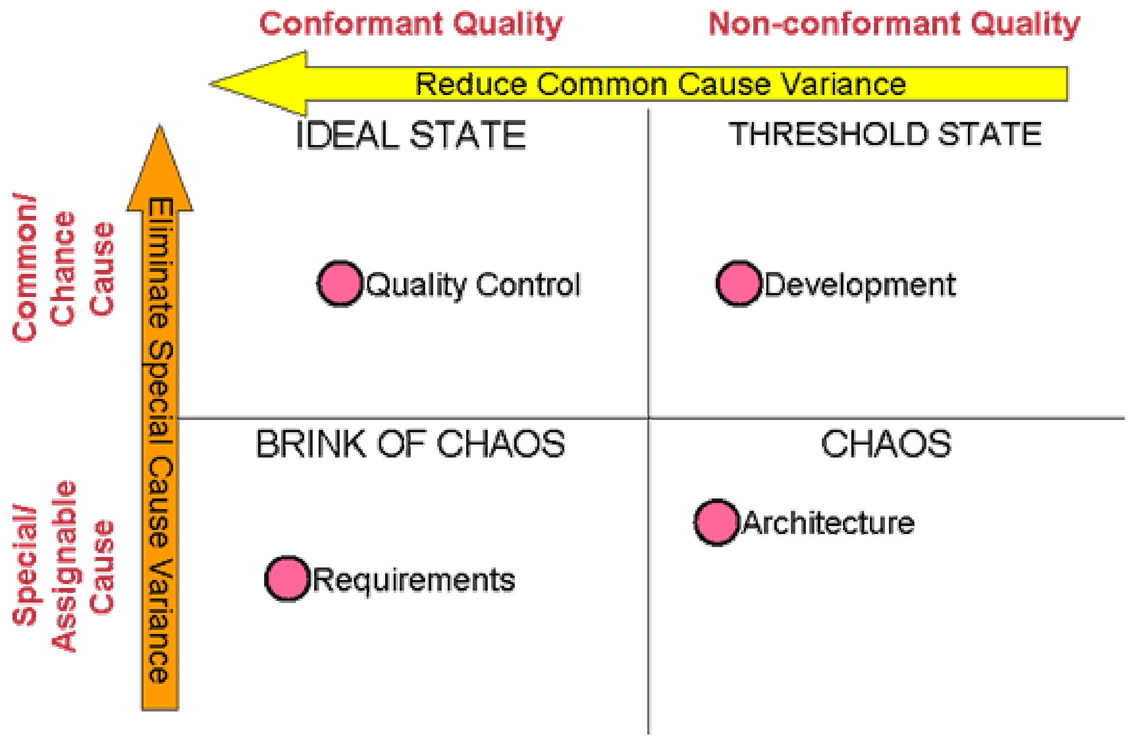


Figure 9. Capabilities Analysis

What figure 9 does not show is the system constraint. It merely shows the instabilities. Six Sigma capabilities analysis cannot be mistaken for identifying the constraint. Random reduction in variation (both special and common cause) will only increase throughput if it accidentally touches the system's capacity constrained resource (CCR).

Control Charts from Cumulative Flow Data

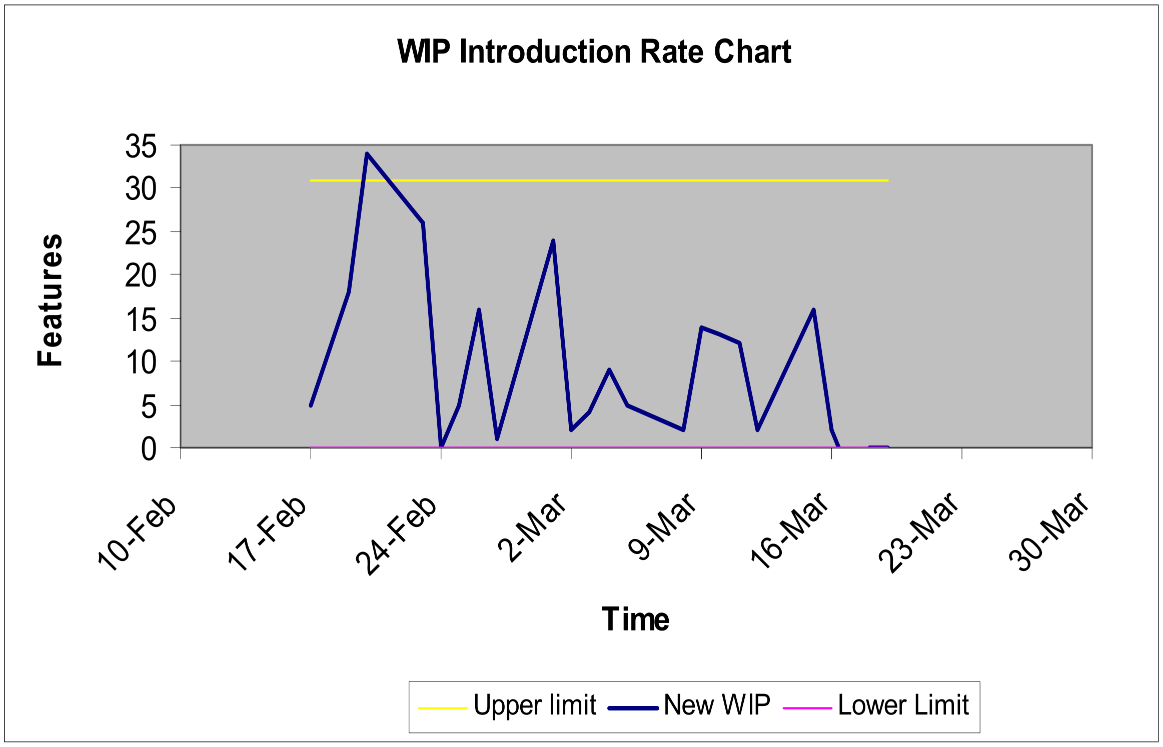


Figure 10. WIP Introduction Rate Chart

Figure 10 charts the introduction of new features as work-in-process taken from the cumulative flow diagram in Figure 6. The calculation of the control limit on the chart is beyond the scope of this paper. This is the rate chart of the WIP Inventory Control Chart, figure 12, and cannot be used in isolation as an introduction rate of 30 features per day is not sustainable.

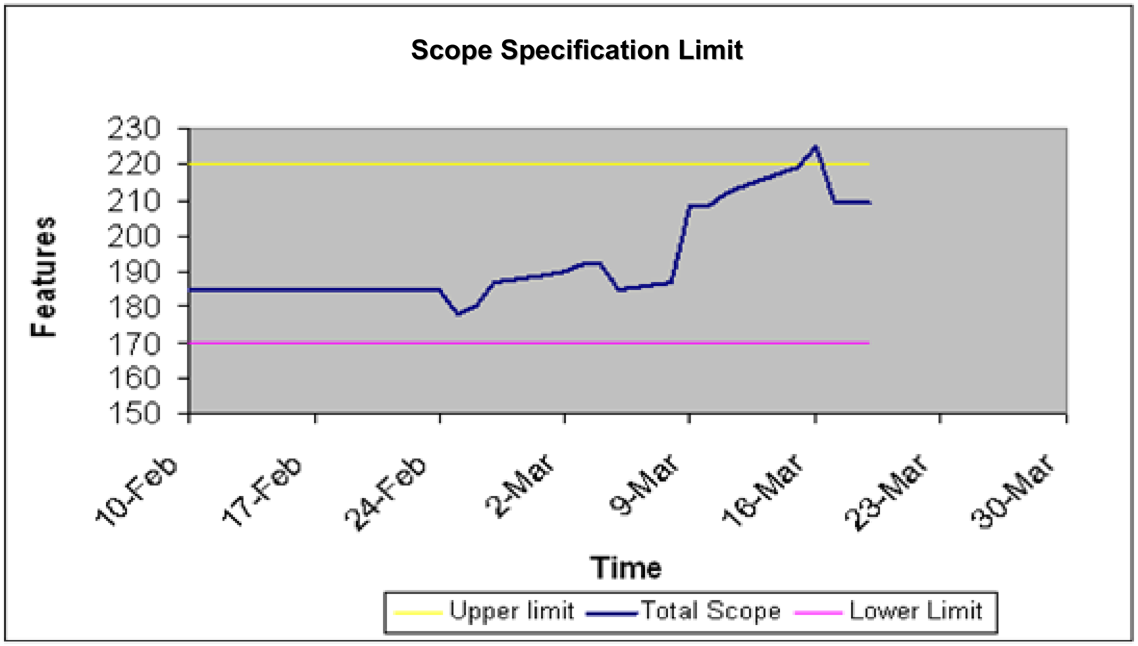


Figure 11. Project Scope Control Chart

Figure 11 charts the capacity buffer usage against the project specification limit – the maximum number of features representing 100% loading on the constraint at the current production rate in the constraint. Extra features come from two sources - project dark matter which is the variation in Features discovered through initial modeling and those that emerge during the project at design time – and, scope creep from change requests which represent new scope for the project.

In figure 11, there are 185 Features in scope at the beginning. If the feature count were to fall below 170 then the manager may decide that it is safe to add some features from the backlog which did not make the prioritization cut during the planning stage. Additional features above 185 represent buffer usage and above 220 the project buffer is consumed in full. This chart cannot be used in isolation to calculate buffer usage, the production rate as plotted on Figure 6 as the rate of features completed must also be used to validate the anticipated delivery date using an appropriate simulation. The second measure checks that the production rate in the constraint is being maintained around the historical mid-point.

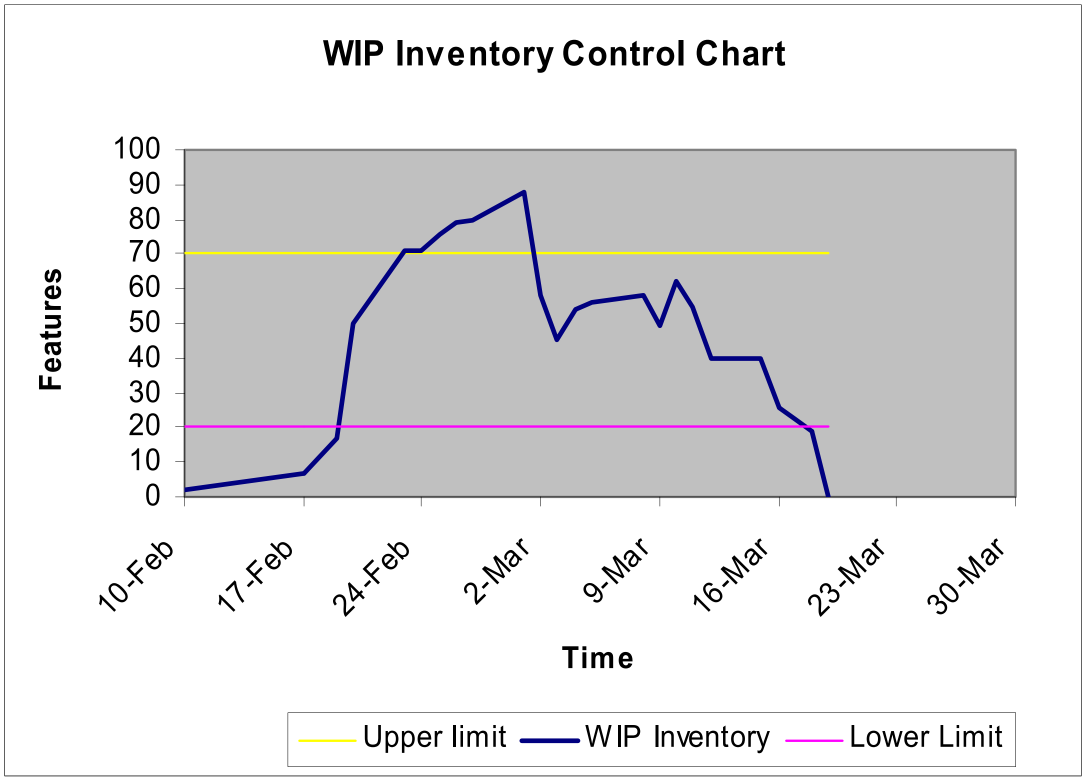


Figure 12. WIP Inventory Control Chart

Figure 12, the WIP Inventory Control Chart shows us how much work is in progress. Again, the calculation of the control limits on the chart is out with the scope of this paper. However, the upper natural process limit indicates whether the aggregate complexity is dangerously high and potentially out of control as well as indicating whether or not the desired lead time of less than two weeks can be maintained. The lower natural process limit indicates that there is insufficient work-in-progress to maintain the desired production rate. The process is either stalling or starving from a lack of upstream material. A lagging trace of lead time can be maintained, as shown in Figure 13, and the data should remain within the control limits of 2 and 14 days providing the WIP Inventory was not permitted to become too large or too small. The lead time control chart is little more than a report card, as it can only be plotted historically. Lead time is a lagging indicator. The WIP Inventory chart should be used for day-to-day control whilst the lead time chart may be a more desirable executive report because it demonstrates the system's ability to deliver.

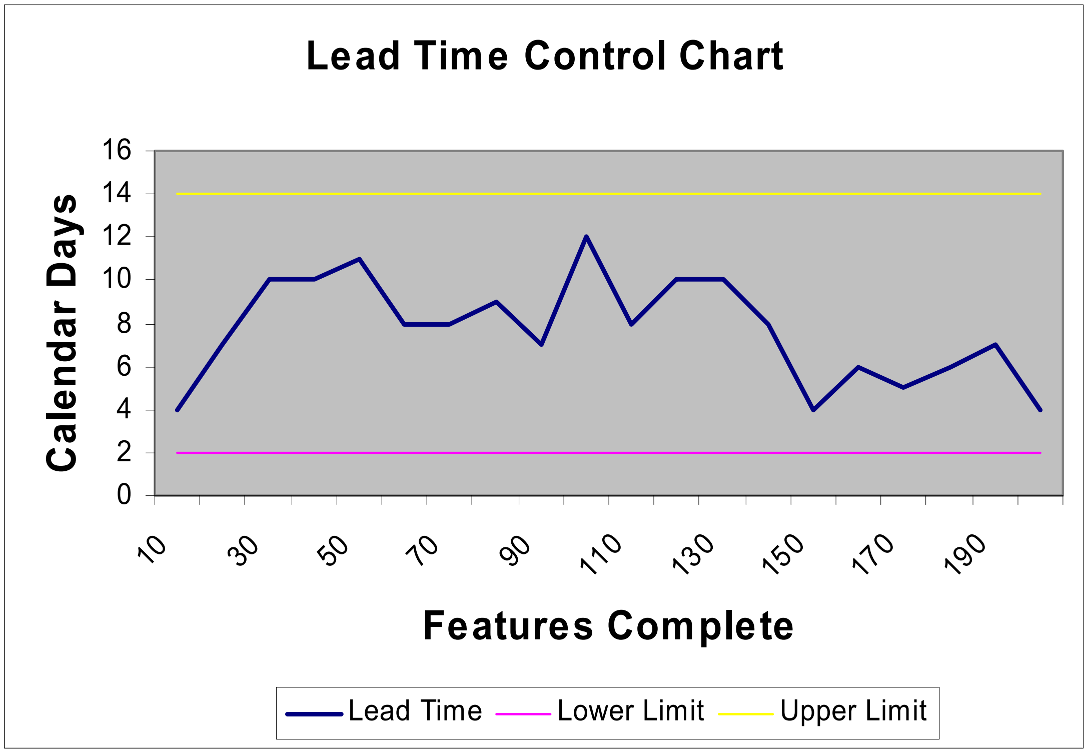


Figure 13. Lead Time Control Chart

Requirements Uncertainty

Figure 14 shows a growing gap between features started but not designed. This is in indicative of uncertainty in the requirements. Such a growing gap in flow, should be echoed by a growing issue log. With features blocked due to insufficient information or ambiguity or lack of internal consistency in the requirements, the development team will start more features with the hope of making progress on those whilst the others are blocked. The overall WIP will grow.

Without a knowledgeable expert on-site customer, it may not be possible to get instant answers. The customer may not even know the correct answer or the business analyst or marketing manager may not be empowered to provide a full answer. Unavailability of subject matter experts (SME's) may cause issues to remain open for long periods resulting in a failure to close the gap. In this respect, a growing gap from started to designed indicates that SME's may be the constraint and/or variation (poor quality) in requirements may be constraining the design function. Regardless, of the reason the cumulative flow plot reveals the symptom and the root cause should be hidden in the issue log. Closing the gap is achieved through effective issue log management, identification of root cause and timely corrective action.

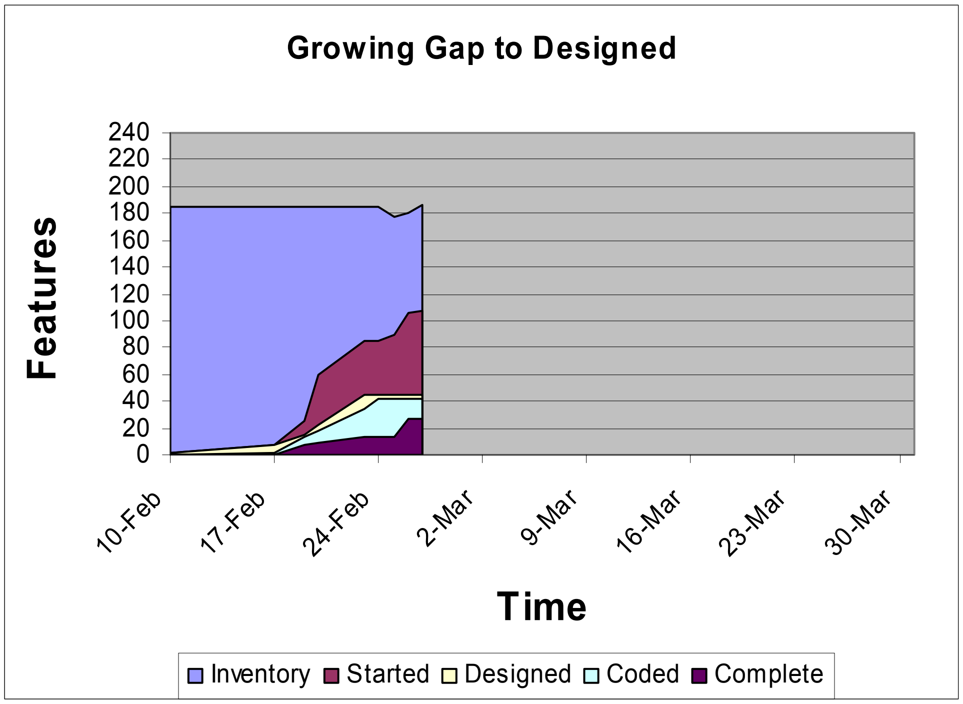


Figure 14. Growing Gap between Started and Designed

Design or Architecture Issues

Figure 15 shows a growing gap between designed and coded. The cause of this cannot be determined from the plot alone but it does provide a red flag which should prompt the manager to investigate.

Poor Design

The first possibility is poor quality of design or unworkable designs. The team declares a design as complete and the chart is updated but when they start coding the design, they discover that they haven't thought it through adequately enough. This means that the real design work is done whilst coding. Hence, the chart was falsely reporting progress. A manager could choose to fix this in two ways – get the team some training in better OO design; or, simply change the reporting method so as not to reflect design. My personal preference is for the former. I believe that good design improves productivity.

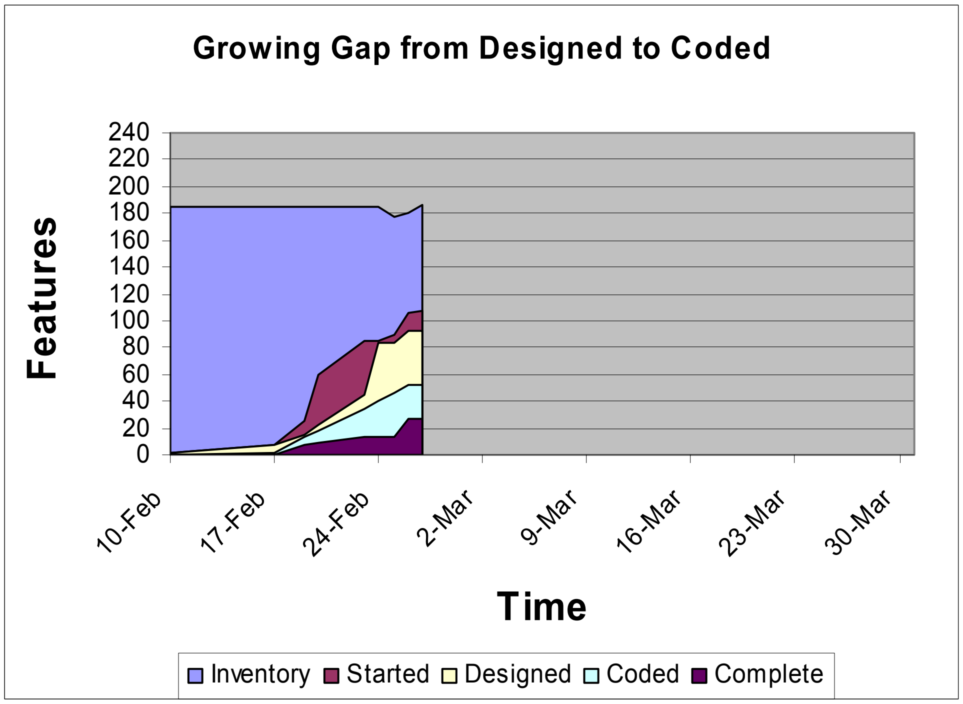


Figure 15. Growing Gap from Designed to Coded

Development Issues

Another reason that the gap from designed to coded could be rising is a lack of knowledge of the development language(s) or the APIs in use. If a new technology is being used for the first time, the team may be struggling to learn it.

Alternatively, there could be environment or tools issues. It isn't unusual to start a project with a temporary license for a tool, platform or environment whilst the procurement people issue a purchase order for the full license. If the temporary license expires then the project is blocked. I have also seen cases where a new project pushes the configuration management environment over the edge and it requires new hardware or a rebuild or reconfiguration. This takes the tools or support team some time and can hold up mainstream development effort.

Refactoring

Undeclared refactoring may be another reason for a growing gap in designed from coded. It may turn out that the latest designs require changes to code written earlier in the project. This rework would not show up as dark matter unless the team were being very honest about it. In an extreme case, it could be that the architecture of the system has been invalidated by the latest feature designs and the whole system is being re-written. This would show as a large gap or a severe downturn in the slope of the "coded" plot on the CFD. [This should not normally happen when Peter Coad's modeling and design method is being followed.]

Just-in-time Domain Analysis

Figure 16 shows a situation where the project inventory is not identified ahead of time but is discovered in a just-in-time fashion.

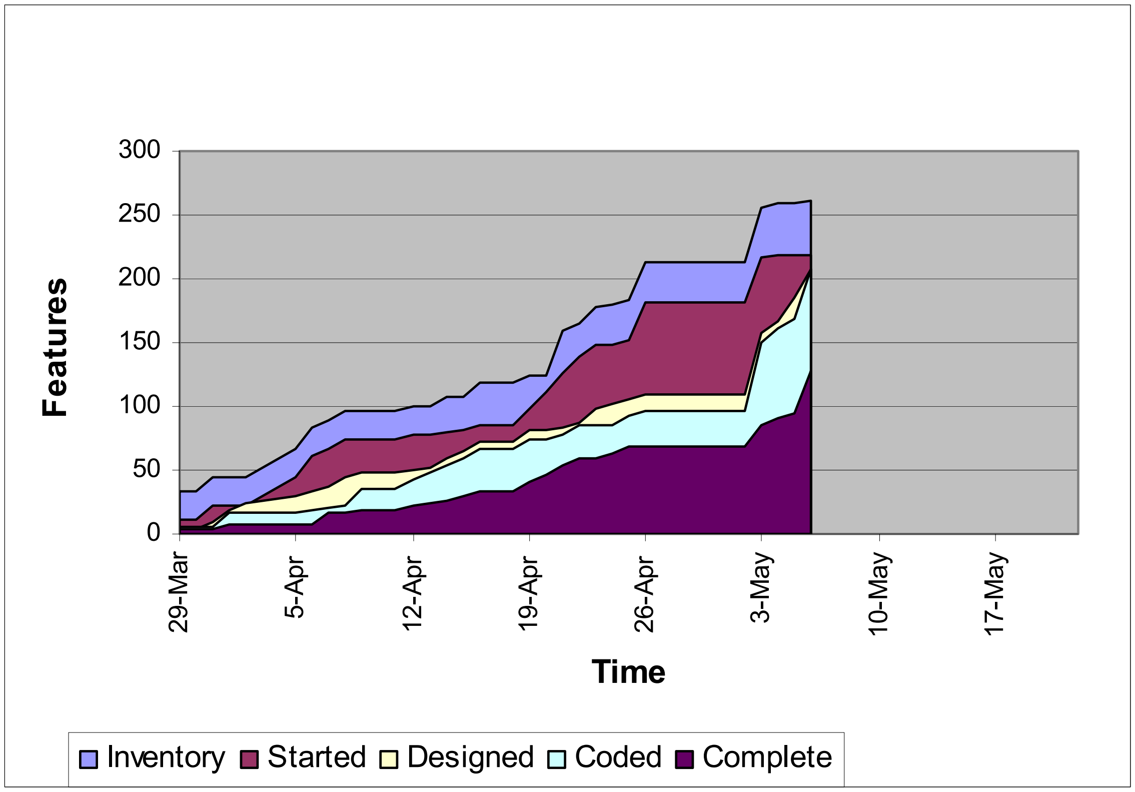


Figure 16 CFD showing just-in-time inventory discovery

Several problems can be uncovered from this picture. Firstly just-in-time analysis is an undesirable effect. The project sponsor and customers hate it. There is no commitment or project promise at the beginning of the project. It is also indicative that there is a constraint – availability of subject matter experts or systems architects (or team leads) who do the domain modeling. The root cause of the constraint on resources – the lack of availability or capacity – may be ambiguity in requirements. In this specific case, the marketing department was churning and vacillating over the product mix choices for an entirely new product category. An equally credible explanation may be that the resources are multi-tasking and are in fact a shared resource. In which case, they should be considered for the synchronizing resource in a Critical Chain multi-project solution (described later).

Monthly Operations Review

Data from CFDs can be used to identify the productivity rates of different elements in the process of software engineering. This data can be presented at an Operations Review. CFDs are intended for day-to-day tactical control use. Operations Review is about learning and organizational feedback. Learning provides input for strategic and operational investment decisions. It provides the material for good governance. Figure 16 shows a process diagram labeled with element capacities per month clearly identifying the current system constraint as System Test. According to the Theory of Constraints, maximum throughput (or productivity) is achieved by exploiting a constrained resource to the full and subordinating everything else in the system to the exploitation decision made. Continuous improvement is achieved by elevating the constraint thus removing it as the system capacity constrained resource (CCR). The effect is to move the constraint elsewhere whilst the system goes on to achieve a higher throughput.

The exploit decision made for this example is: all system testers have been relieved of all non-value-adding tasks. The subordination decisions made are: extra project managers have been assigned to complete those non-value-adding administrative tasks; system analysts have been assigned to write test plans for a future release, freeing up testers to work purely on current tests; and new material flow into the system has been restricted to 100 units per quarter, from a demand of 100 units per month. The elevation decision made to remove the constraint is to hire 5 temporary staff to relieve the bottleneck.

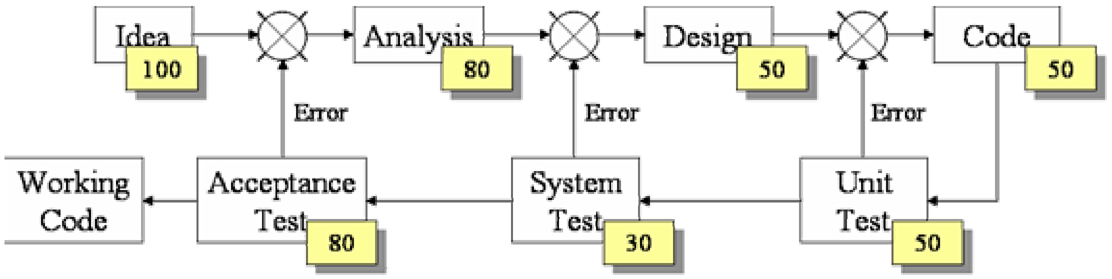


Figure 16. Process Flow and Capacity

Operations Review should also report the status of all projects in a simplified rollup which provides accurate information properly aligned with the organizational goals. Figure 17 shows a buffer management parking lot diagram for a larger project with 7 Subject Areas. The colors on the chart indicate the overall buffer usage for the subject areas. This simple chart demonstrates clearly the software engineering organization's ability to meet its promises with respect to wider programs and the dependent dates. The information is entirely objective and is derived from the rollup of fine-grained ideas for client-valued functionality, expressed as Features, laid out in a buffer protected Critical Chain schedule with an understanding of the intrinsic variability in the method. Figure 17 provides objective transparency of properly aligned engineering activities.

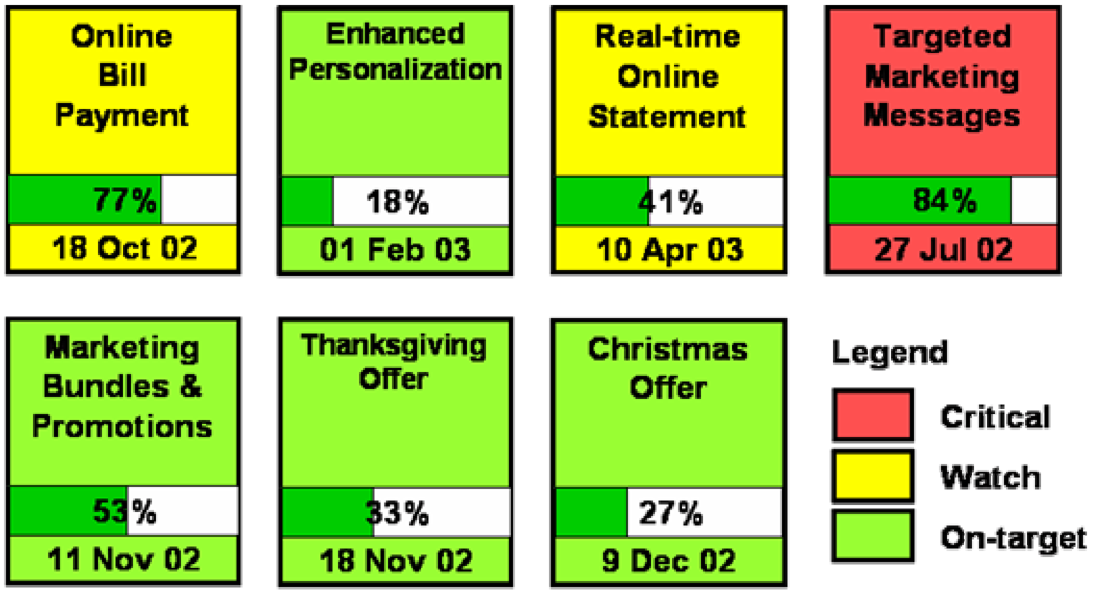
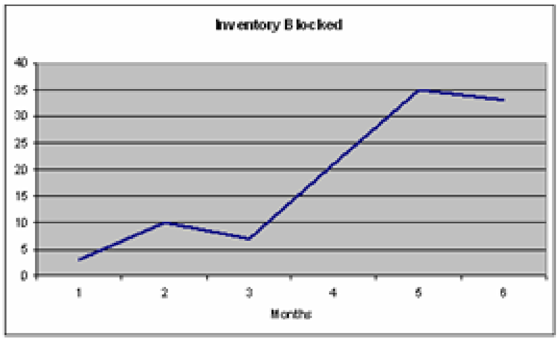


Figure 17. Buffer Management Parking Lot Chart

Executives at Operations Review are likely to focus their time on problem areas. In the example of Figure 17, Targeted Marketing Messages is shown as Critical status. A drill down of detail for Targeted Marketing Messages is shown in Figure 18. Only a few data points are required. The Subject Area is broken out into the feeding chains of Feature Sets and buffers for each feeding chain and the overall project buffer (treating the Subject Area as a sub-project in itself) is shown. This provides a finer granularity in to the area causing concern. In addition the number of units of inventory (Features) blocked are shown along with the recent trend, and the number of open issues in the issue log causing those blockages along with the recent trend.

Targeted Marketing Messages
84%
27 Jul 02

	Percentage Used	Days Remaining
Project Buffer	90%	2
Feeding Buffer 1	150%	0
Feeding Buffer 2	3%	17



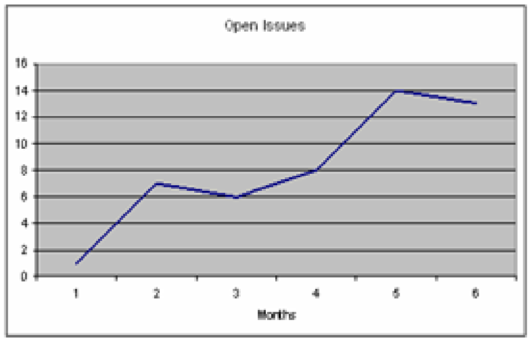


Figure 18. Subject Area Drill Down

The open issues represent what is constraining the productivity and completion of the blocked Features. The Theory of Constraints teaches us to focus on the constraint. Hence, all management effort must be applied to eliminating the blocking issues in order to restore flow and bring the errant Subject Area back under control. If simply unblocking issues is not enough to recover the buffer then management may have to decide how to elevate in order to recover or to re-plan based on a new baseline for delivery.

Maximizing Throughput

Throughput is difficult to define in advance. Marketing projections are at best a black art. However, market research and competitive strategy [Porter 1980] can be used to estimate the size of a market, the likely sales price and the market share achieved. Hence, estimates can be made. Michael McGrath [1995] provided a framework for market segmentation which allows us to make a fine grained assessment of the Features in a product definition which will appeal and enable particular market segments. Throughput Accounting would suggest that all throughput generated from a sale should be assigned to the enabling differentiating features which swung the purchase decision [Anderson 2003]. Figure 19 provides a simple example of features for an FDD knowledge base tool which segments the market by prospective user role and by price per client application for those prospective roles. Firstly, the feature list is grouped into sets of features which apply against the market segmentation as shown in Figure 19a. Figure 19b then maps these feature groups against the role and price segmentation. Figure 19c maps the anticipated throughput generated from each feature group in each sector.

Requirement / Release	Group	Rating
List the Features for the Project/Release	1	Essential
Set the Milestones for a Feature	1	Essential
Set the CPW for the Feature	1	Essential
Set the Subject Area for the Feature Set	1	Essential
List the Feature Completion Dates for the CPW	1	Essential
List the Virtual Team members for the CPW	1	Essential
Set the Feature Set for the Feature	1	Essential
List the Feature Completion Dates for the Release	2	Essential
Total the Features for a Product	2	Essential
List all Feature Sets for the Subject Area	3	Desired
List the Subject Areas for the Product	3	Desired
List Change Requests for the Release	4	Optional
Total the number of open issues in the Issue Log for a given Release	4	Optional

Figure 19a. Feature List grouped by market segment

Free				
$0 - $500	Feature Group 4			
$500 - $1,000				
> $1,000		Feature Group 1 Group 2	Feature Group 2 Group 3	
Price per Seat / User Role	Project Mgr	Development Mgr	Program Mgr	Product Mgr

Figure 19b. Feature groups by Market Segmentation

Free				
$0 - $500	$10,000			
$500 - $1,000				
> $1,000		$70,000 $30,000	$30,000 $20,000	
Price per Seat / User Role	Project Mgr	Development Mgr	Program Mgr	Product Mgr

Figure 19c. Anticipated Throughput by Market Segment

Product Mix Prioritization

This anticipated Throughput data can be made to make product mix decisions and allocate resources in the most optimal way providing the system constraint is already identified.

Throughput Accounting [Corbett 1997] simply says that we must sort the product mix by the anticipated Throughput over the utilization of the current constraint. Figure 20 shows the data from Figure 19 mapped against the estimated utilization of the CCR – the System Test function.

Requirement / Release	Group	# Man Hours to Test	Group Total Testing	Throughput	$ / hour of CCR
List the Feature Completion Dates for the Release	2	32			
Total the Features for a Product	2	32	64	$60,000	$937.50
List the Features for the Project/Release	1	8			
Set the Milestones for a Feature	1	8			
Set the CPW for the Feature	1	16			
Set the Subject Area for the Feature Set	1	20			
List the Feature Completion Dates for the CPW	1	16			
List the Virtual Team members for the CPW	1	8			
Set the Feature Set for the Feature	1	8	100	$70,000	$700
List all Feature Sets for the Subject Area	3	16			
List the Subject Areas for the Product	3	20	36	$20,000	$555.56
List Change Requests for the Release	4	64			
Total the number of open issues in the Issue Log for a given Release	4	64	128	$10,000	$78.13

Figure 20. Feature Group sorted by Throughput/CCR utilization

This data shows us that Feature Group 2 is most likely to generate the best return with Feature Group 4 delivering the least return. As resources are constrained and only so many features can be delivered in the next release, this data allows the product manager to make an informed decision not to enter the market segment for project managers with the next product release.

The Throughput Accounting product mix solution is in this case being used to replace the Kano modeling technique described earlier.

Critical Chain

We can enable a Critical Chain solution based around the feature concept and its inherent variation. The Feature Set should be selected as the task unit for the Critical Chain schedule. A PERT is created showing the dependencies between the Feature Sets.

User Interface (UI) Feature Sets will have dependencies on back-end code or business logic. Figure 22 shows a CCPM PERT chart for a small development project.

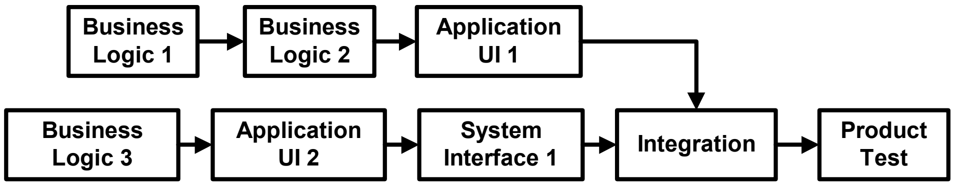


Figure 22. Critical Chain PERT Chart for a Software Project

This initial PERT chart has no buffers to protect the delivery date. In order to protect the delivery date a project buffer must be introduced. The size of the project buffer is based on the unprotected lead time estimate for the Critical Path – in the case of Figure 22 that would be Business Logic 3 through Product Test. The size of the feeding buffer is based on the lead time estimate for the feeding chain of Business Logic 1 through Application UI 1. Lead time estimates are based on historical data for production of Features. The buffer size is based on the empirically observed variation in Feature lead time. Figure 23 shows a revised PERT chart with buffers introduced.

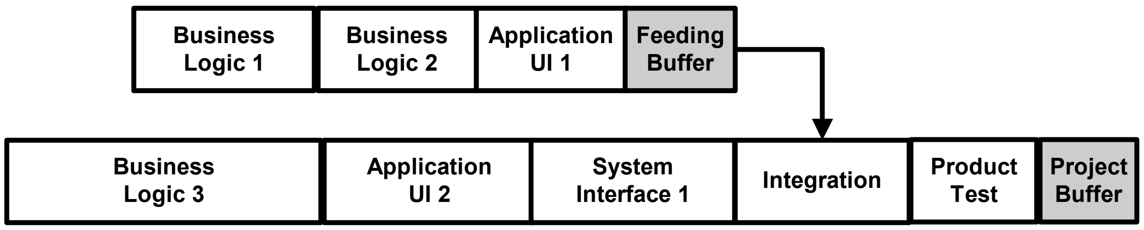


Figure 23. Protected Critical Chain PERT Chart for a Software Project

This plan is much better as the delivery date is now protected by buffers which will absorb variance during the development cycle. However, the plan does not yet include the user interface (UI) design activity. It is important to add UI design activity to the plan because application UI coding is dependent upon it and the UI Design group is a shared resource. We can choose to make this shared resource the system CCR (or synchronizer). Hence, the plan has to be revised to add the UI design activity and to buffer it appropriately. If UI design is not buffered to protect it from uncertainty then UI design resources may not be available when they are needed and any delay will eat away at the Project Buffer. Figure 24 shows the completed plan with the UI Design team resource scheduled into the project.

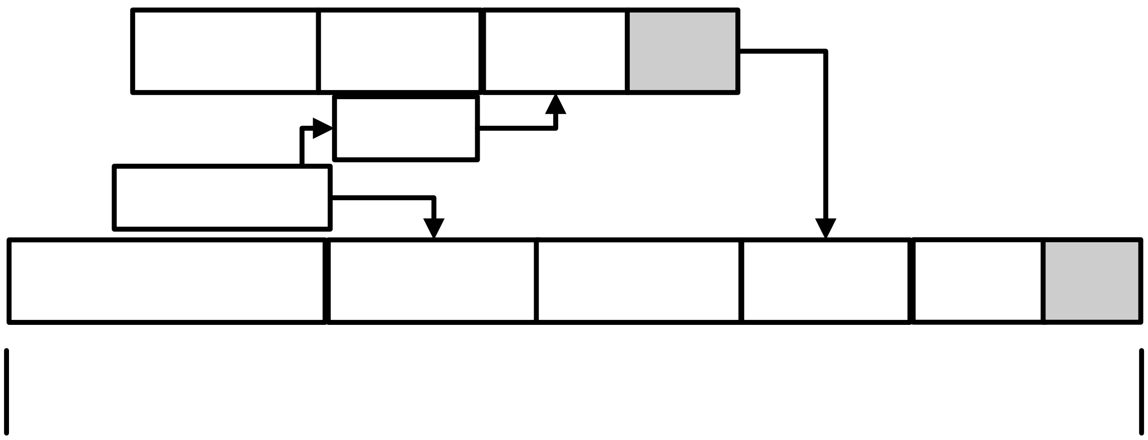


Figure 24. Final Critical Chain Project Plan with UI Design as CCR

Multi-Project (Program Management) Solution

We can synchronize a program of projects through the shared UI design resources. When the UI Design group is chosen as the CCR, the flow of UI design work – development of detailed user interface specifications can be used to synchronize all work activities. All other functions including software development and quality assurance will incur slack time because by definition they will never be entirely loaded.

The work of the UI Design team, as the system synchronizer, will now need to be prioritized based on the order of precedence of the projects under way. Each project must be prioritized. This could be done based on Throughput value, or delivery date, or strategic value to the business, potential market share gains, or defensive posturing against an aggressive competitor. Regardless of the method used, the business owners must agree to prioritize the projects in development.

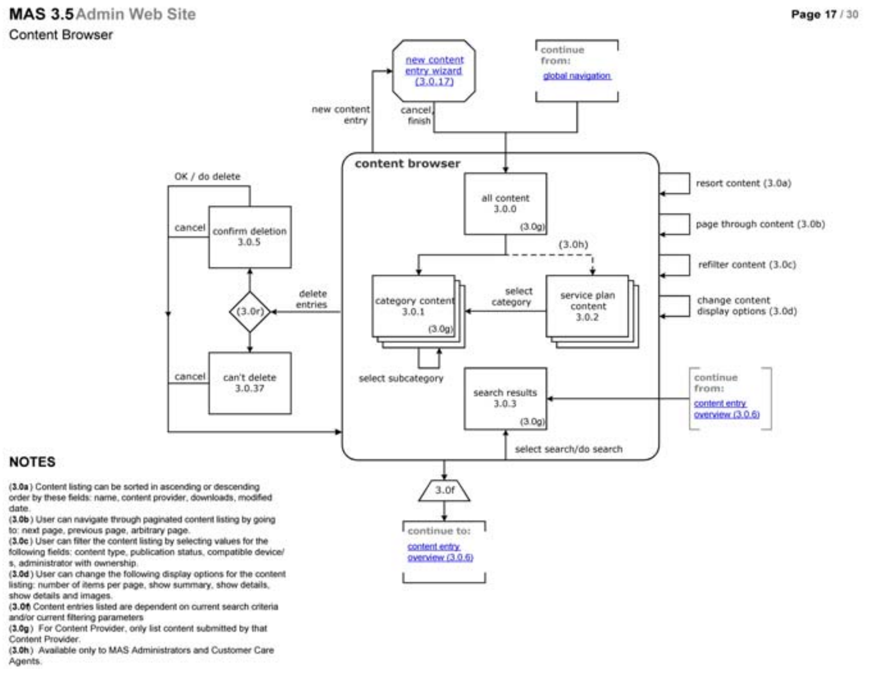
The release of work for each project is now staggered based on the capacity and availability of the synchronizing resources – the UI designers. When there is a conflict, the higher priority project takes precedence and the lower priority one is re-scheduled.

In order to insure that UI Design resources are always available when promised, a capacity buffer is introduced whenever a UI Design resource is switched from one project to another. The capacity buffer is designed to absorb uncertainty in the UI design process and insure that the UI designer is always available to start on the next project by the promised date.

Scheduling UI Design as the Drum

The criteria for a synchronizing drum resource are that it should be fairly close to the beginning of the system so that the rope (and its associated variation) are small and that the synchronizer exhibits very low variation. Low variation insures that it remains the system constraint as designed. High variation may cause the constraint to move uncontrollably. This is undesirable.

In order to enable UI design as a synchronizing resource, it is necessary to bring UI design under control and reduce its variation. To do this a modeling technique was required for planning UI design activity and deliverables. A survey of available techniques finally settled on the use of Visual Vocabulary [Garrett 2002, 2002] as a suitable technique. Figure 25 shows an example of a Visual Vocabulary model for a user interface design.



NOTES

(**3.0a**) Content listing can be sorted in ascending or descending order by these fields: name, content provider, downloads, modified date.
(**3.0b**) User can navigate through paginated content listing by going to: next page, previous page, arbitrary page.
(**3.0c**) User can filter the content listing by selecting values for the following fields: content type, publication status, compatible device/s, administrator with ownership.
(**3.0d**) User can change the following display options for the content listing: number of items per page, show summary, show details, show details and images.
(**3.0f**) Content entries listed are dependent on current search criteria and/or current filtering parameters
(**3.0g**) For Content Provider, only list content submitted by that Content Provider.
(**3.0h**) Available only to MAS Administrators and Customer Care Agents.

Figure 25. VV Model for MAS Content Browser (courtesy 4thpass Inc.)

The UI Design manager needs the UI design effort estimates from the model and the project prioritization in order to schedule the UI resources. The CCPM plans for each project provide an indication of the earliest time the UI resources are required and the duration they will be needed. The UI resources must be scheduled in order to best facilitate the desired project delivery dates whilst not exceeding the buffered capacity of the UI Design team. Figure 26 shows a revised plan for a two project multi-project organization using UI Design as the synchronizing system CCR.

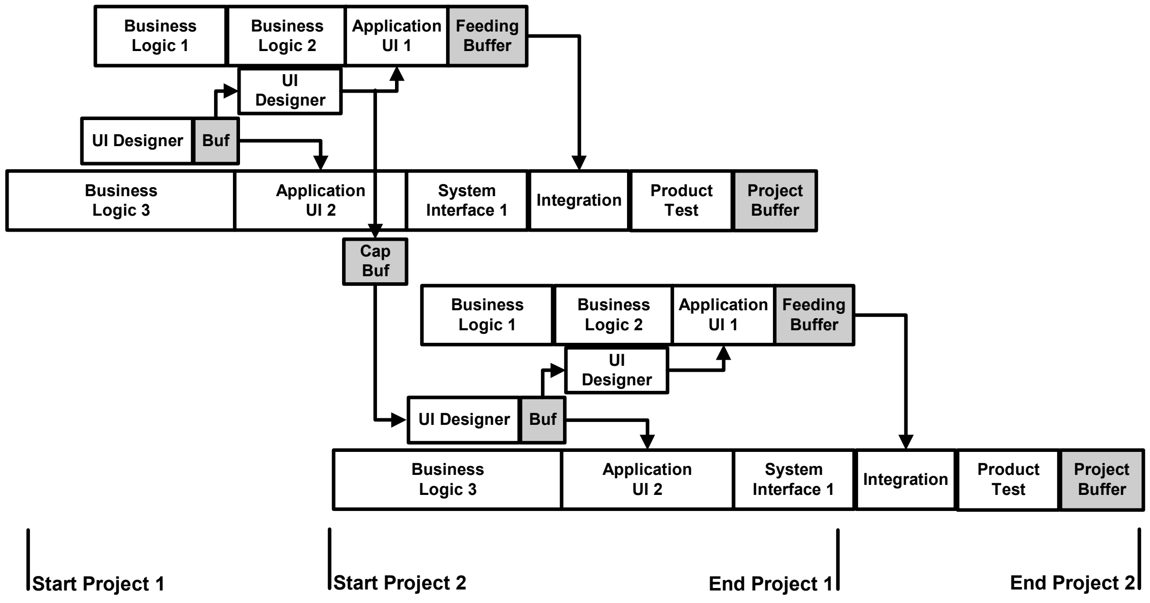


Figure 26. A Multi-Project Plan with UI Design as CCR showing Capacity Buffer

Implications of Fully Exploiting UI Design as the CCR

By definition the CCR in a multi-project CCPM solution must be fully exploited. This means that it will be fully loaded to its capacity with the exception of the capacity buffer. Hence, the UI designers will be the busiest and most focused professionals in the organization. Whilst others will experience slack time which may facilitate undertaking of tools development, bench projects, self-learning or formal training, the UI design team, on the other hand, will be fully loaded.

It may therefore be necessary to supplement the UI Design team, from time to time, with temporary contract workers, so that permanent staff can be rotated off for training and professional growth.

Summary

The Coad Method and Feature Driven Development through the novel definition of a Feature as a fine-grained piece of consumer-valued functionality which can be directly mapped to design is the core enabler for Theory of Constraints, Lean and Six Sigma management techniques in software engineering.

The Drum-Buffer-Rope solution in TOC can be used with Feature Driven Development. Cumulative Flow Diagrams (from Lean Production) which map the flow of value creation through the FDD transformative milestones can be used to identify the bottlenecks in the process. Derivative data from the CFDs can be used as input for statistical process control charts which can be used to enable both a Six Sigma and a Theory of Profound Knowledge (Deming) understanding of the capabilities in our system and its inherent variation.

Understanding of variation allows us to make Critical Chain plans based on objective tolerance data for lead time in FDD Feature production. A multi-project solution can be achieved by designating a shared resource such as user interface design as the synchronizing constraint. Variation in the synchronizer can be reduced by introducing a modeling method which identifies "value items" and permits a framework for measuring production lead time in the synchronizer. For user interface design, the use of Visual Vocabulary diagrams has met this requirement.

The Throughput Accounting product mix solution can be used to select the Feature mix for an FDD project and is an effective replacement for the Kano modeling technique previously deployed in the method.

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 works in Redmond Washington as a Program Manager with Microsoft. 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 publishes 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 Professional Technical Reference, 2003
[Boehm 1981] Boehm, Barry W., ***Software Engineering Economics***, Prentice Hall, 1981
[Coad 1999] Coad, Peter, Eric Le Febvre and Jeff De Luca, ***Java Modeling in Color with UML: Entreprise Components and Process***, Prentice Hall PTR, 1999
[Cunningham 2001] Beck, Kent, Beedle, Mike, van Bennekum, Arie, Cockburn, Alistair, Cunningham, Ward, Fowler, Martin, Grenning, James, Highsmith, Jim, Hunt, Andrew, Jeffries, Ron, Kern, Jon, Marick, Brian, Martin, Robert C., Mellor, Steve, Schwaber, Ken, Sutherland, Jeff, Thomas, Dave, ***Manifesto for Agile Software Development***, 2001
[Deming 2000] Deming, W. Edwards, ***Out of Chrisis***, MIT Press, 2000
[Garrett 2000] Garrett, Jesse James, ***A Visual Vocabulary for Describing Information Architecture and Information Design***, http://www.jjg.net/ia/visvocab/
[Garrett 2002] Garrett, Jesse James, ***The Elements of User Experience: User Centered Design for the Web***, New Riders, 2002
[Goldratt 1984] Goldratt, Eliyahu M., ***The Goal***, The North River Press, 1984
[Goldratt 1997] Goldratt, Eliyahu M., ***Critical Chain***, The North River Press, 1997
[Leach 2005] Leach, Lawrence P., ***Critical Chain Project Management***, 2nd Edition, Artech House, 2005
[McConnell 1998], McConnell, Steve***, Software Project Survival Guide***, Microsoft Press, 1998
[Palmer 2002] Palmer, Stephen R., Felsing, John M., ***A Practical Guide to Feature Driven Development***, Prentice Hall, 2002
[Patterson 1992] Patterson Marvin, Accelerating Innovation – Improving the Process of Product Development, Von Nostrand Reinhold 1992
[Poppendieck 2003] Poppendieck, Mary and Poppendieck, Tom, ***Lean Software Development: An Agile Toolkit***, Addison Wesley, 2003
[Reinertsen 1997] Reinertsen, Donald G., ***Managing the Design Factory – A Product Developer's Toolkit***, Free Press, 1997
[Wheeler 1992] Wheeler, Donald J., and David S. Chambers, ***Understanding Statistical Process Control***, SPC Press, 1992