The Theory of Constraints

Productivity Metrics in Software Development

Photo by aimee_n. Licensed under Creative Commons.

Introduction

This paper is largely based on the work of David J. Anderson, in "Agile Management For Software Engineering". It also includes some of my own interpretations and understandings of the Theory of Constraints. The original intent of this paper was to facilitate the discussion of productivity and metrics in the Development Department at McLane Advanced Technologies, LLC. This paper is not intended to be a comprehensive or exhaustive discussion of the points within, but it intended to spur additional research and conversations. I hope you enjoy reading it as much as I enjoyed writing it.

Theory of Constraints

The core of the Theory of Constrains (TOC) is that for any given system, at any given time, there will be at least one constraint on that system, determining how quickly the system can produce.

> *TOC, applied to manufacturing, seeks to identify bottlenecks in the production line. The underlying assumption is that a production facility is only as fast as the slowest process in the chain. As a general rule, TOC assumes that a value chain is only as strong as the weakest link in the chain. The capacity of the weakest link is the current system constraint.*
>
> - David Anderson, "Agile Management for Software Engineering"[1]

TOC is a systems approach that looks at every part of a system, from concept to cash. Individual steps are not considered to be of highest priority – that is, TOC does not optimize a single step in a system to ensure that this single step is working to its full capacity. Instead, TOC's primary focus is to maximize the throughput of a system. Given the knowledge of the system's constraints, it seeks to increase the production capabilities of that system, while reducing the cost of production and shortening the time needed to produce.

There are five basic steps outlined by TOC, to accomplish these goals:

1. Identify the constraint(s)
2. Exploit the constraint to maximize productivity
3. Subordinate all other steps or processes to the speed or capacity of the constraint
4. Elevate the constraint – in other words, work to remove the current constraint, leading to higher capacity or production rate for the entire system
5. Lather, rinse, repeat. ☺

Although TOC was originally a system for improving the production capabilities of manufacturing, it can be applied to other industries – including software development. David Anderson's book is essentially a 300+ page case study on how Sprint.com operates their software development efforts and other related processes and services.

Throughput Accounting

Throughput Accounting is also a systems approach – it examines the entire cost structure of a system, from concept to cash. This is in direct opposition to standard cost-accounting systems, which seek to maximize the production rate of single steps in a given system, with no regard for the production capacity of the entire system. The goal of applying Throughput Accounting with TOC is to increase the throughput of the system while decreasing the inventory and investment, and decreasing the operating expenses of that system.

At this point, there is no real need to distinguish between TOC and Throughput Accounting. Therefore, they will both be referred to as TOC.

Metrics in TOC

There are a few high level categories of metrics that can be found in TOC: Production metrics and Financial metrics. Production metrics help us to measure how much of what we are producing and how quickly we are producing it. The financial metrics use the production metrics to tell us the cost and profit of the software that we are producing.

For simplicity in explanations, I am going to assume that a single feature or end-user function is how we are tracking the development efforts. This is only one way of tracking, though. There is additional discussion on applying the metrics of TOC to various methodologies, later.

Production Metrics

The production metrics measured by TOC include

- Inventory (V)
- Lead Time (LT)
- Quantity: Units of Production (Q)
- Production Rate (R)

Inventory (V)

Inventory in software development is not as straight forward as manufacturing. What are the “raw materials” or “unfinished goods” that we need to track as inventory? What this comes down to, is how we track productivity through our system. Anything that moves through our system in considered inventory when it is not actively being worked on. This includes any ideas that we have for features and functionality in the system, and unfinished work currently waiting on something – people to work on it, answers to questions, etc, and even finished work that is not yet delivered to the customer.

The cost of inventory storage in software development is negligible. We use source control and document management systems to store this inventory, reducing the storage cost down to almost nothing. However, the intellectual cost of inventory is significant. When the system puts aside unfinished work for a period of time, there is a minimum amount of effort that is required to re-asses where the work was left and whether or not the existing design and functionality is still valuable and correct. Any unfinished work that is no longer valid for any reason is re-work (all re-work is waste), and the cost incurred during the production of the unfinished work must be written off as Operational Expense (OE - discussed later).

Quantity: Unit of Production (Q)

This is the total number of units of production that have moved through the software development system, to date. Q can be incremented at the time a feature or function is complete, that is "done, done, done"[2] – ready to be delivered to the customer. However, any feature or function that is not delivered to the customer yet, is considered inventory. If a feature is done and the customer then decides that they do not want the feature, the cost of producing that feature must be written off as Operational Expense, and Q is not incremented for the feature.

Lead Time (LT)

Lead time is the time it takes to move a feature from initial idea, through the system, delivered to the customer. That is, the time it takes to product one unit of production (Q).

Production Rate (PR)

Production rate is the number of units (Q) produced during a given period of time. That is, how many features or functions were developed to completion during a given period of time – 3 units per week; 2 units per month; etc.

Optimizing Lead Time (LT) vs. Production Rate (PR)

It is preferable to focus improvement efforts on the Lead Time of a software development system, rather than the Production Rate. A decrease in Lead Time will increase the Production Rate. However, focusing on Production Rate does not always decrease Lead Time. The net effect of these changes – positive or negative – has a corresponding affect to the financial metrics of the development system.

Focusing on Lead Time allows a software development team to rapidly produce the features and functions that a customer wants, as soon as they ask for them. A team may only produce 4 units (Q) per month - but if the Production Rate is 1 per week, then the customer is receiving updates on a weekly basis. As the software development system begins to shorten its Lead Time for feature development, the Production Rate will increase. When a team's Lead Time drops from 1 unit per week (5 days) down to 1 unit in 3 days, the Production Rate will have increased from 4 units per month to around 6.67 units per month. That's an increase in Production Rate of 59% while simultaneously delivering the features more frequently.

Focusing on Production Rate may lead to improvements in Lead Time as well. However, it may also lead to a team that has a larger inventory stacked up in the development system. That is, a team may be producing 10 units (Q) per month, but they may take 3 months to move each of the features from concept to release. The longer a feature sits in the system, the more that feature will cost – the intellectual cost of having to re-asses the feature's state and restart the push through the system.

In order to decrease the Lead Time of the system, we must reduce the Inventory currently in the system and also remove the waste in the system. The reduction of Inventory allows a team to focus on what the customer currently needs. This corresponds to a reduction in Operational Expense by reducing the amount of time and effort that is spent re-learning a feature or function that has been sitting in inventory. The reduction of waste in the system allows the team to produce the features faster by eliminating the waste in the system – the problems that cause delays. The system can produce software faster, thereby reducing the Operational Expense again.

Financial Metrics

Other metrics, such as Return On Investment and Net Profit can be derived from the production metrics.

- Investment (I)
- Operating Expense (OE)
- Throughput (T)
- Net Profit (NP) = T – OE
- Return On Investment (ROI) = NP / I
- Average Cost Per Function (ACPF)

Investment (I)

Investment is the total dollar amount in a given software development system.

> Investment "is all money invested in software production systems plus money spent to obtain ideas for client-valued functionality. It thus includes software development tools and requirements gathering. For a one-time project, however, its I is not constant, so I is subtracted from T at the end."
>
> - John Arthur Rickkets, "Reaching The Goal"[3]

The investment in a system of software development has a direct impact on the Return On Investment for the software being developed.

Operating Expense (OE)

Operating expense is the cost of converting an idea into working software. This includes salaries of team any over-head categories of labor in a company. For example, the Business Development department who originally won the contract, the HR department to staff the team, the IT department to support the network and email for the team, the physical housing costs of the building, etc.

All waste – that is, activity or work that produces no value in the eyes of the customer – is part of Operational Expense. This includes rework and code that is thrown away because the customer does not need it or because it has bugs and needs to be cleaned up or fixed.

Throughput (T)

Throughput is the dollar amount of a feature sold, minus the cost of installing or delivering the feature. This is typically measured as a Throughput rate – how much money is earned per unit of production (Q).

Maximizing Net Profit (NP) and Return On Investment (ROI)

> “the way to maximize NP and ROI in [Throughput Accounting] is to increase T while decreasing I and OE. This is done in software engineering by gathering requirements rapidly yet accurately, creating software that customers value, and eliminating waste, which is composed of requirements and functions that are discarded before software enters production.”
>
> – John Arthur Ricketts, “Reaching The Goal”[3]

Average Cost Per Function (ACPF)

The cost of software development efforts (new features in existing systems, new systems in general, or even bug fixes in existing systems) can be calculated as the Average Cost per Function. This is the dollar amount that the software development system will spend to create an individual feature.

- Average Cost Per Function (ACPF) = OE / Q

Additional Metrics for the Business of Software Development

There are additional metrics for monitoring the business of software development, with TOC. Both David Anderson and John Ricketts have outlined these metrics, how to calculate them and how to apply them, in their respective books:

- Agile Management for Software Engineering, by David J. Anderson [1]
- Reaching The Goal, by John Arthur Ricketts [3]

Applying TOC to Software Development

Software development is not a "production" system, but is a system of design and development. As such, we cannot accurately predict the final size, shape, complexity, or a number of other aspects of the systems that we are creating. We can, however, measure these aspects of existing systems as they are produced. We can measure the emergence of the aspects of our system and use that empirical knowledge to give a range of expected results for the given team, working on new features or functionality.

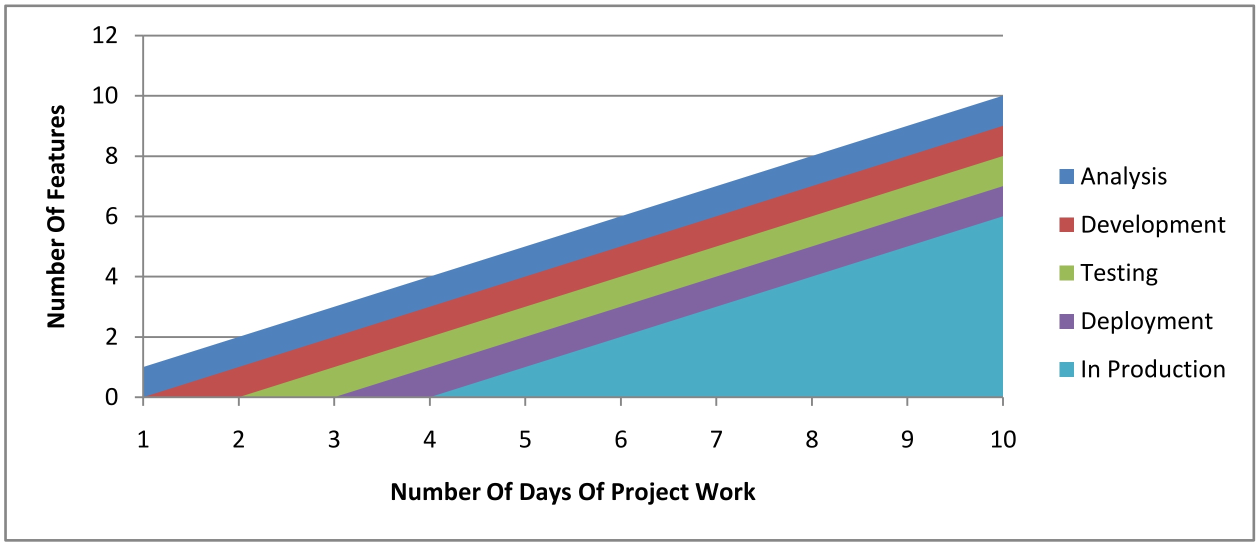
Estimating Software Development

Given the Average Cost Per Function, and the known average Lead Time for functions, software estimation becomes significantly easier. Rather than estimating the amount of time that a team believes a feature or function will take, they only need to estimate the size of the features and functions. Once the sizes and quantities of features and functions are known, multiply those quantities by the Average Cost Per Functions for the estimated cost of the system. Then multiple those quantities by average Lead Time to get the total estimate for the time it will take to create the system, feature requests, or bug fixes in question.

Monitoring Productivity

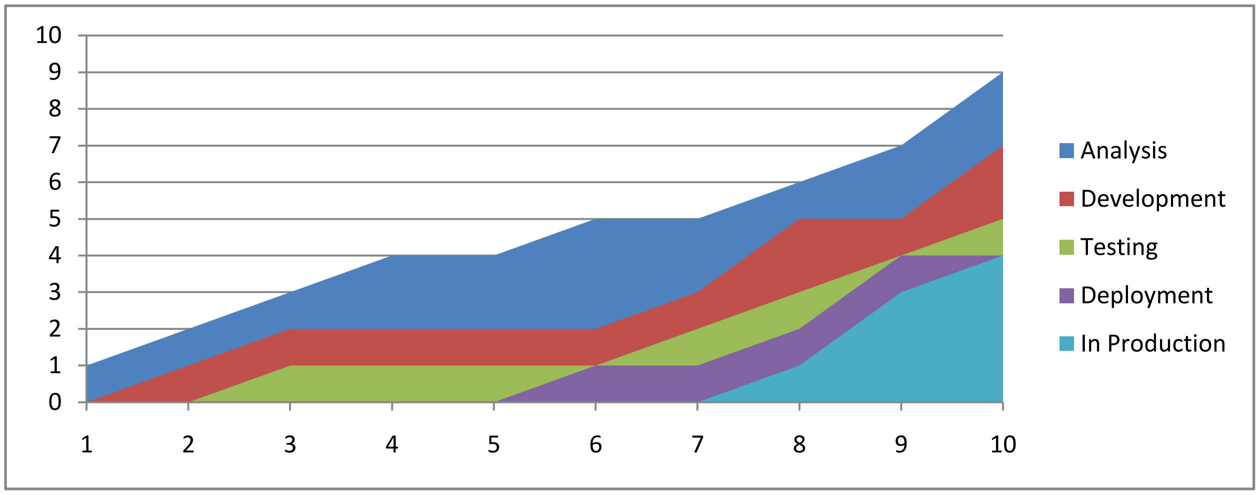
To monitor the productivity of a feature or function through the system, TOC often borrows the "Cumulative Flow Diagram" (CFD) from Lean thinking. This type of diagram illustrates how the features or functions are flowing through the system.

For example, if we have a perfectly stable system where one feature is requested every day, and one feature is delivered every 5 days, our CFD would look like this:



This shows that a single feature travelled from Analysis, through the system, and was deployed in 5 days (Lead Time).

Of course a more realistic world would not be quite so clean cut. Many different features may take many different time frames. A closer-to-reality example of a CFD may look like this:



In this chart, we can see that there is a much different flow of the features through the system. The analysis process appears to be occuring much faster than the rest of the system can keep up with. The development effort may actually be the constraint on the system, according to this diagram. Essentially, when a flat spot is found in the flow, that is a signal that features are becoming stuck, they have run into problems, or are taking longer than normal to be processed.

Defining Metrics For Methodologies

When we attempt to apply the metrics from TOC into an environment other than manufacturing, the definition of those metrics becomes important. After all, what is a “Unit Of Production (Q)” or “Inventory (I)” in software development? They are certainly not the same as manufacturing – for example, a Unit Of Production for a cell phone manufacturer would be a finished cell phone, and inventory would be the sub-assemblies (LCD screens, keyboards, radios) and other parts that go into the final products.

The metrics that we choose to insert into these definitions need to meet a few key requirements:

- They should be self generating – that is, we should not have to run complex calculations or be involved in any great amount of process to create the metrics from our systems process.
- They should reflect our goals – that is, we should choose metrics that directly correlate to the principles and processes of our company. The metrics we choose should support the financial numbers that the department and company want, as well as the productivity and quality numbers that we want.

- They should be leading or predictive indicators of how the system will perform in the future.

The good news in this somewhat ambiguous situation is that we can apply TOC to a wide variety of project types, teams, and customer needs. There are some clear definitions of each metric that TOC wants to use, in David Anderson's "Agile Management" [1] book, for various styles of software development. Some of the examples described, include:

SDLC ("waterfall" Software Development Lifecycle)

- Inventory: Function Points (as defined by IFPUG – International Function Point Users Group) noted in functional specifications
- Investment: cost of creating functional specifications and performing Function Point Analysis
- Lead Time: The time it takes to move a Function Point through design, development, testing, and delivery to the customer
- Throughput: the dollar value added from the delivered Function Points
- Production Rate: the number of Function Points delivered in a given time period
- And others…

Extreme Programming (XP)

- Inventory: story points
- Throughput: the dollar value of a delivered story point
- Production Rate: the "velocity" of the team – average number of story points per iteration
- Investment: the cost of acquiring user stories
- Lead Time: the length of time to deliver a story to a customer – usually one iteration length, possibly 2 or more iteration lengths if a team is staggering development and testing
- And others

Others

Other types of development and management styles that are covered, include FDD (Feature Driven Development), Scrum and RAD (Rapid Application Development).

References

[1] David Anderson, “Agile Management for Software Engineering”
http://www.amazon.com/Agile-Management-Software-Engineering-Constraints/dp/0131424602

[2] Jeremy Miller, “"Code Complete" is a lie, "Done, done, done" is the truth”
http://codebetter.com/blogs/jeremy.miller/archive/2006/04/13/142800.aspx

[3] John Arthur Rickkets, “Reaching The Goal”
http://www.amazon.com/Reaching-Goal-Managers-Goldratts-Constraints/dp/0132333120

Additional Sources

David J. Anderson: Agile Management
http://agilemanagement.net/

Eliyahu Goldratt, “The Goal”
http://www.amazon.com/Goal-Process-Ongoing-Improvement/dp/0884271781

Wikipedia
http://en.wikipedia.org/wiki/Theory_of_constraints

Google ☺
http://lmgtfy.com/?q=theory+of+constraints+in+software+development

Contact Information

Derick Bailey is a professional software engineer and architect, working for McLane Advanced Technologies, LLC. He is a blogger with the LosTechies community, and active in the Austin area developer community and beyond.

Please feel free to contact me regarding this, or any other papers and articles that I have written and published.

- Email: derick {at sign} derickbailey {period} com
- Blog: http://derickbailey.lostechies.com
- Twitter: @derickbailey

Copyright Notice and Licensing