TOCICO is pleased to recognize this white paper as part of the TOC Body of Knowledge.

“Agile” CCPM:
Critical Chain for Software Development

Koichi Ujigawa

Director of TOC System Development Division, Being Co., Ltd.

Japan

David Updegrove

Theory of Constraints Consultant, Owner, Clearview Solutions

705 SW Oak Road

Port Orchard, WA 98367

United States of America

TOCICO White Paper Series

Editor: Victoria J Mabin, PhD
Professor of Management, Victoria University of Wellington, New Zealand
Vicky.mabin@vuw.ac.nz

Table of Contents

Table of Figures

Executive Summary

This paper suggests expanding the current TOC knowledge in Critical Chain Project Management (CCPM) by examining a modified network construction process and buffer management method for use in software development. The problem we are seeking to address is the assumption held by many people in software development that CCPM, and by extension TOC, may not offer an acceptable (or good enough) solution for their project management challenges. Therefore, they embrace other methodologies, predominately Agile, as their preferred solution.

Agile project management was introduced as a response to the perceived unreasonable demands of traditional waterfall project management within the context of software development, allowing developers more adaptability, responsiveness, and autonomy over their projects, and creating for them a much more positive work atmosphere. While being a win for developers, Agile is often perceived as a capitulation for the greater organization, who still must manage the customer imperatives of cost, schedule, and scope.

Critical Chain Project Management has repeatedly proven to be an effective method for achieving these customer imperatives and meeting the needs of the overall organization. It is being well-received and is growing rapidly in many project environments. But when introduced in many software development environments, Agile project management tends to prevail because of its perceived benefits for the development team – benefits that such teams will not relinquish willingly.

“Agile” CCPM is designed to create a win-win between developers and their parent organizations via a new proposed set of techniques that allow developers the adaptability, responsiveness and autonomy they desire while giving the larger organization the ability to better manage commitments to customers. While some techniques are borrowed from the Agile and pre-Agile worlds, the solution is fully compliant with the fundamental principles of CCPM and the Theory of Constraints, making “Agile” CCPM a customization of CCPM, as opposed to an entirely new or hybrid methodology.

Background

In many typical software development situations, the traditional waterfall project management method mandated for meeting strict schedule commitments is felt to be intolerable – for example where project task descriptions have not been fully identified, task duration estimates are thought to be highly speculative or even unknowable, and when drastic changes[1] in a project's work content can be experienced in a very short period. To make matters worse, the traditional waterfall method has not been shown to consistently meet such commitments in the first place.[2] Developers therefore often feel they are under great pressure to provide results they believe are impossible to achieve. This has been the primary driver behind finding a different, more workable methodology for project management in software development.

Agile project management has been developed over the past 20 years specifically as a response to this problem. The purpose of Agile methodology is to 1) quickly adapt to changes in software project scope, 2) provide superior responsiveness to those generating the changes, 3) achieve greater speed of execution than traditional methods, and 4) achieve continuous value delivery. Since Agile is believed to be much more adaptable to change than waterfall, and by using sprints and other Agile tools, development teams can be viewed as delivering something of continuous value (both early in and throughout the project) – from the developers' and often the customer's perspective, Agile is usually seen as superior to waterfall.

In addition, and the importance of this must not be overlooked, Agile methodology offers a level of autonomy to the development team that is not present in waterfall.[3]

This autonomy allows development teams greater control over the delivery aspects of projects.[4] It has given developers something they consider to be of great personal value, and thus they wish to retain it. This benefit triad of adaptability, responsiveness and autonomy has

---

[1] In addition to regular change orders, these changes can be driven by new devices such as smart phones and tablets, progressive infrastructures such as cloud services (SaaS, PaaS, IaaS, etc.) and innovations in the development/programming and operating system environments. In other words, even the rules of the game can change rapidly.

[2] According to a Gartner report roughly at the time where CCPM and Agile solutions began to gain traction (2000), "Roughly 40% of all IT projects fail to meet business requirements": http://www.techrepublic.com/article/it-project-failures-costly-techrepublic-gartner-study-finds/

[3] See the Agile Manifesto: http://www.agilemanifesto.org/

[4] Note that one of the key values of the Agile Manifesto is: "Responding to change (takes priority over) following a plan."

become for software developers what would be referred to, in the terminology from Goldratt's Change Matrix[5], a huge *mermaid[6],* or something that is meeting present needs or desires, giving up on which would be a perceived as extremely negative.

However, while Agile may help meet the needs of developers in relation to individual projects, and while there are attempts (such as SAFe[7]) at making Agile more compatible with multiple projects being in execution concurrently, Agile does not necessarily meet the needs of the *company* in terms of project portfolio management, or the needs of its customers, who are often demanding firm commitments on schedule, cost, and scope. This forces Agile organizations either into a less than satisfactory compromise, or into giving up completely on the predictability of firm commitments. Perhaps worse, some Agile techniques—such as time-boxing work into sprints—carry the undesirable consequence of creating frequent stops in the flow of task completions, and often interrupt task owners in the middle of their tasks. Worse still, Agile portfolio management frameworks such as SAFe require the application of Agile for all projects, even those for which it may be ill-suited, without providing a systematic means for improving the reliable throughput of project completions.

Critical Chain Project Management (CCPM) has been developed with the intent of overcoming the problems of traditional project management (projects finish late, projects finish over budget, and quality and/or scope of projects is routinely compromised). CCPM is intended for many divergent project environments, including extremely complicated multi-project environments. It also addresses the entire project portfolio and the holistic projects organization. When properly implemented, CCPM has been shown to consistently succeed in overcoming these problems through the utilization of properly defined and constructed project networks, properly sized and buffered task duration estimates, buffer management to provide early warning of potential problems, and a smooth, balanced and sustainable load on the system via project staggering.

In our experience, when introduced to CCPM, software developers will often acknowledge its advantages over traditional project management. Some will even try it. However, for many, in the end they perceive the focus on delivery and the need for task duration estimates as ultimately being closer to the waterfall model – in their minds an attempt to force certainty on uncertainty. It needs to be recognized that there is always a psychological fear of being forced into commitments in situations where one does not hold all the cards. CCPM is sometimes

[5] "Do people resist change? Isn't it obvious?" YouTube video: https://www.youtube.com/watch?v=hcz1aZ60k7w
[6] For additional information on "mermaids" and Goldratt's Change Matrix, see Barnard, Alan, *The Change Matrix Cloud Process,* Draft White Paper, TOCICO White Paper Series, 2016.
[7] SAFe is an acronym for Scaled Agile Framework® www.scaledagileframework.com

misunderstood to the extent that it is presumed to be a continuation of the traditional waterfall mindset of estimation and commitment.

The desire for adaptability, responsiveness and autonomy by the development team often trumps the benefits offered by CCPM, and they turn to (or continue to embrace) Agile. This is not so much a rejection of CCPM as it is a reluctance to give up their mermaid. Mermaids aren't inherently bad or unworthy. They are the positive of not changing. Therefore, in a solution, it is much more persuasive to show that they get to keep their mermaids – win-win.

In choosing the perceived execution benefits of Agile over the planning benefits of waterfall, they unknowingly reject an opportunity for even greater improvement than is available from *either* waterfall or Agile – an opportunity that is offered by CCPM.

Can CCPM meet the needs of developers?

In response to Agile project management, some TOC practitioners have innovated – incorporating various Agile or Lean methods into CCPM, and have achieved some admirable success, with some developers seeing this as delivering value from both worlds. However, in doing this the methodologies sometimes overlap one another, are not always implemented consistently between practitioners, are not currently part of the accepted TOC BOK, and have the potential for obfuscating the TOC solution within the project management marketplace.

This paper describes an alternative approach for obtaining the adaptability, responsiveness and autonomy of Agile via CCPM – an approach which relies on changes in the way we arrive at task duration estimates and how this corresponds to the project's buffers. The new techniques proposed are fully consistent with TOC philosophy, can be used with existing CCPM software, and for simplicity and ease of use can be incorporated into existing and future CCPM software products. Earlier versions of the proposed solution have been presented at the 2012 – 2014 TOCICO conferences (see references).

It is significant to note that in contrast to pure Agile methodology, "sprints", or arbitrarily defined fixed "time boxes" are not required with the proposed method. In fact, such fixed time boxes can be counter-productive since they *are* fixed. If one has overestimated the size of the sprint, they should finish early and move on. If it is underestimated, one should continue until the objective is met. Therefore, sprints act as milestones, removing flexibility from the project, and they encourage negative behaviors such as Parkinson's Law.

Removing sprints from the solution is an advantage in terms of minimizing project duration, controlling scope creep, and utilizing buffer management to achieve consistent on-time delivery. Since the project team is working to complete story points (defined below) and individual features, the crucial adaptability in developing each new feature and the completed end product is retained.

This eliminates the major reason that organizations reject CCPM in favor of Agile.

The format for presenting the "Agile" CCPM solution is based on the Standing-On-The-Shoulders-Of-Giants (SOG) process introduced by Dr. Eli Goldratt at TOCICO in 2011.

SOG Step 1 – Identify a giant

This white paper stands on the shoulders of Dr. Goldratt and the TOC application Critical Chain Project Management. CCPM has successfully addressed the significant shortcomings of traditional project management[8] in many project environments, including in our experience major assembly, engineering to order (ETO), construction, Maintenance, Overhaul & Repair (MRO), new product development (NPD) and software development.

SOG Step 2 – Identify the enormity of the area not addressed by the giant

As stated above, in our experience (and in that of many colleagues and client organizations) CCPM is an effective solution for the problems plaguing project organizations in many divergent work environments. However, particularly in software development, CCPM is perceived by

[8] Including claims that projects are often late, over budget, and/or must reduce scope to accommodate schedule or cost; lack proper planning, lack communication, and lack synchronization, especially in multi-project organizations.

some as being deficient, or at least the superiority of standard[9] CCPM cannot be clearly claimed. Since we have accumulated personal experience with marketing and implementation of CCPM in software development, the solution outlined in this white paper addresses this industry specifically.

Development of new software or new features within existing software is often subject to one or more of the following attributes:

- Characteristics / features are not deterministic and are non-linear
- Major changes often occur during execution
- New ideas can emerge through project execution
- The sequence in which things needs to be done can be very flexible (success may sometimes be sensitive to sequence)

Although the process steps for creating new software features may be well-defined, due to the above attributes the time it takes to execute these steps is highly variable. When asked for task duration estimates, a common response in software development is, "We have no idea." Furthermore, it is often felt that the task descriptions themselves (and therefore the level of effort) cannot be fully determined until they are discovered in execution.

The pressure and mandate at company and portfolio levels to create precise project networks (which may even have to be pre-approved by the customer), and to meet pre-determined completion dates creates resistance to the idea of detailed planning, numeric task duration estimates and project completion dates. This resistance has been a primary impetus for the development of Agile project management.

Since using established CCPM implementation techniques typically involves solid project networks, clearly defined task descriptions and absolute[10] task duration estimates, this resistance often surfaces, even when we insist that variability is accounted for in the buffers and estimates are not commitments. For software developers, such requests recall the pain encountered in waterfall. It can be extremely difficult to complete the paradigm shift that CCPM is adaptable, responsive, and liberating enough for software developers to consistently choose it over Agile.

[9] For the purposes of this paper, "standard" CCPM is defined as the present widely-accepted form of CCPM where task duration estimates are gathered as absolute numbers (see footnote 10) and buffers are managed in relation to the extent of consumption of these absolute-numbers in aggregation from all tasks in a chain.

[10] 'Absolute' means we are asking for a single, real number, even if this number is understood not to be a commitment and is protected by buffers.

The current reality for many TOC implementers and practitioners is that CCPM is not being adopted as extensively as it should be in IT environments, and we consider this as an area not (adequately) addressed by the giant of CCPM.

It is important to note that we are not claiming that CCPM does not work in software development or is incapable of handling such scenarios. On the contrary, we have personal experience of success in CCPM for software development. However, we do claim that the traditional implementation techniques of CCPM are not necessarily the most effective techniques for addressing the factors listed above, and that the established CCPM approach is empirically not effective in changing the perception that Agile is a better (more credible and effective) solution for the management of software development organizations.

SOG Step 3 – Get on the giant's shoulders

CCPM is a tested and proven method to consistently deliver on three valuable advantages for projects organizations – on-time delivery, within-budget performance, and maintenance of scope. Since these strengths bring so much value to a projects organization in terms of reputation, sales, and competitive advantage, they should not be relinquished easily.

When the considered use of CCPM is abandoned by organizations in lieu of Agile or other methods, it is a tacit admission that on-time delivery, within-budget performance, and scope maintenance are for practical purposes impossible – it is a surrender of sorts. We feel that such capitulation is not only unnecessary, it also contributes to the organization's failure to realize its potential in terms of profit, reputation, and competitive advantage.

Therefore, in the proposed application enhancement, we climb upon the shoulders of the full established CCPM application and begin from there. All the benefits and attributes of CCPM are maintained while building in the adaptability, responsiveness and autonomy of some of the practices adopted by Agile. None of the fundamental rules or benefits of CCPM are sacrificed.

For this reason, the application enhancement is truly CCPM. It is agile (small "a"), not in the sense that it is a hybridization of Agile and CCPM, but in the sense that it makes standard CCPM more adaptable to change. Other ways to describe this method could be "Responsive", "Adaptive", or "Flexible" CCPM. But for clarity and concept recognition among those familiar with the software development community, we have titled this method "Agile CCPM."

SOG Step 4 – Identify the conceptual difference

Projects are undertakings designed to produce a specific product, service or result.[11] They are temporary or transitory in nature, having a discernable beginning and end. Although they may to varying degrees share some characteristics with various forms of manufacturing or production, projects can also differ from these in several ways.

Whereas manufacturing tends to be highly repetitive, with production runs often consisting of hundreds or thousands of identical items, projects are often non-repetitive, producing unique end items or results. In situations where projects are repetitive and produce fewer discrete end items, even then the end items tend to have greater customization between them.

Manufacturing also tends to have lower process variability in its operations steps, and the individual steps usually require minimal management attention. Projects, on the other hand, tend to contain steps (or tasks) with much higher variability and uncertainty, sometimes requiring significant management attention when problems arise. Finally, in manufacturing, touch time is normally a small percentage of lead time (the largest contributor to lead time is the waiting time *between* tasks), whereas in projects touch time tends to be very high relative to project lead time, and the preponderance of wait time occurs *within* the tasks.

Some organizations offer products or services which inhabit a "gray area" between manufacturing and projects. These are situations where although "touch time" is high, project networks are very "flat," i.e. there are few feeding paths and convergence points, projects tend to be quite repetitive, and variability tends to be low. This situation seems to be best addressed by the High-Touch-Time Drum-Buffer-Rope (HTT-DBR) solution[12].

There are also distinctions within the realm of projects. In most project environments, *what needs to be done* (at least at a conceptual level) is well known. A clearly-defined project network can be built showing sequences of dependent tasks, even if the specific project has not been executed before. Individual tasks within projects have often previously been executed

[11] For the Project Management Institute (PMI) definition of a project, see: http://www.pmi.org/About-Us/About-Us-What-is-Project-Management.aspx

[12] http://www.tocico.net/CORRECTED-Scheinkopf-Kishira-Schragenheim%20Foundation%20-%20Final%202012-0521_2012-09-08.pdf

multiple times, providing experience for workers and managers and allowing “reasonable” task duration estimates.

Many project environments generally meet these criteria. This includes many engineering environments, major assembly, and construction. CCPM was originally conceived to address such an environment: the design and construction of offshore oil rigs. Therefore, the conceptual foundation of established CCPM is based around these criteria. The current practice of building critical chain plans and buffer management are an outcome of this conceptual foundation.

However, in software development, this is very rarely the case. In many projects one knows quite little about the detailed content of future tasks, and developers can often only recognize the relative *size* of tasks and their approximate order. The *what must be don*e in the project is less clear, sometimes far less so. For example, it is often the case that the customer does not have a complete picture of what they want until a working preliminary software is provided. Sometimes, unexpected (fatal) defects can be caused by a small change made during programming, occurring at a large distance from the change. Finding those in advance is often extremely difficult.[13]

Finally, in software development, an even semi-confident quotation of reliable task duration estimates is much more difficult. For many, Absolute Time Estimation (ATE) is an exercise in futility.

For many software engineers, this makes the accepted CCPM project network construction process more problematic, going so far as to allow some to challenge its validity for software development.

To gain greater acceptance of CCPM in software development, a new conceptual framework is proposed, where Relative Size Estimation (RSE) is more reliable than Absolute Time Estimation[14], and velocity-based buffer management is a superior way to measure project health and bring projects in on-time, within budget, and with better maintenance of quality and scope. These can be accomplished within the true spirit and logical boundaries of TOC and CCPM, and will be addressed in detail below.

[13] Although this can also happen in large engineering projects, it is far more common in software development

[14] For example, “new product/function X seems larger (takes longer to develop) than product/function Y”.

SOG Step 5 – Identify the wrong assumption

The growing number of CCPM success stories make it easy for inertia to help us form a belief that the current techniques of CCPM implementation are the most effective techniques for all project types and all project environments. It is always tempting to believe that because we were successful in one project environment "we know" that if we implement CCPM in the same way we will be similarly successful the next time. Therefore, we run the risk of overlooking significant differences in the characteristics of different project organizations. In their book, "**The Invisible Gorilla, And Other Ways Our Intuitions Deceive Us**", research psychologists Christopher Chabris and Daniel Simons refer to this as a cognitive bias called the "Illusion of Confidence."[15] This is not a criticism of the professionals who implement CCPM. Rather, it is merely human nature.

A similar cognitive bias can cause us to believe that the *techniques* we use in implementing a solution are sacred and unalterable. Therefore, we run the risk of believing that although some minor accommodations can be made for different situations and product or service types, overall, CCPM is a fixed set of unchangeable techniques. We are prone to *not* revisit and challenge the assumptions behind our techniques. Again, this is human nature. Chabris and Simons refer to this as the "Illusion of Knowledge."[16]

Of course, we are warned in the Five Focusing Steps about using methods based on assumptions where those assumptions are not valid. For every new situation, assumptions can and should be challenged. We believe that for software development there is a benefit in challenging the following assumption in current CCPM practice:

It is necessary to estimate* a priori *the time required for every task in a project network.

Due to the conditions that often exist in software development as described in SOG Step 4 above, we claim this assumption is not always correct, especially when creating new software products or making major upgrades to existing products. Therefore, some of the current techniques used in project network construction and time-based buffer management would need to be changed if CCPM is to be perceived as superior and established as a best practice for more and more software development organizations.

---

[15] Chabris, Christopher, and Simons, Daniel, 2009, *The Invisible Gorilla: And Other Ways Our Intuitions Deceive Us,* Crown Publishing Group, New York, NY

[16] Ibid.

SOG Step 6 – Conduct the full analysis to determine the core problem, solution, etc.

Diagnosing the core problem

The pressure to perform well vis-à-vis the imperatives of project management – delivery, cost, scope, quality – exist in software development as much as they do in any other project environment, especially at the company level. However, the nature of software development, i.e. the attributes stated in Step 2 above, makes realizing these imperatives seem much harder to obtain.[17]

On the other hand, Agile project management promises great adaptability and continuous value delivery. This creates an intense conflict between the traditional waterfall project management, and the newer and rapidly expanding field of Agile project management, represented as an Evaporating Cloud[18] in Figure 1:

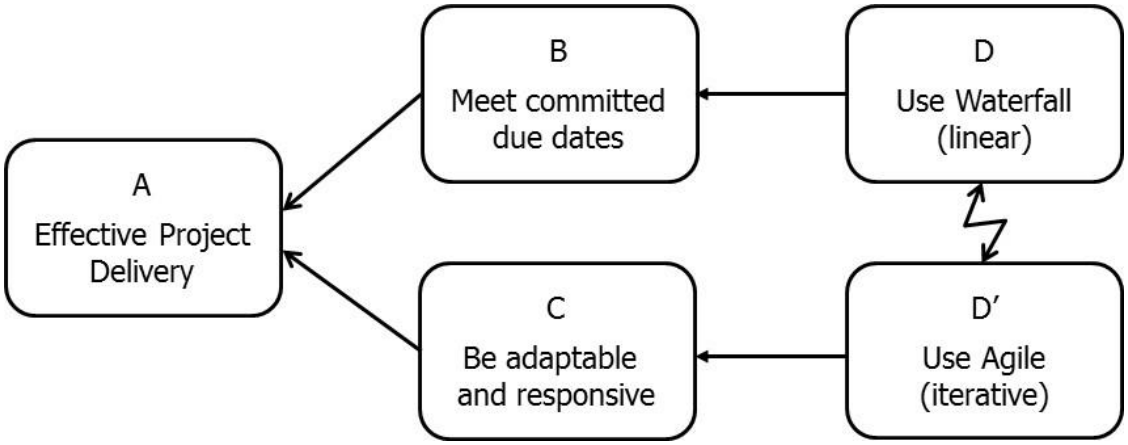

Figure 1 A cloud with CCPM ramifications often present in software development organizations

[17] This is not to say that such pressures do not exist in non-software development environments. Rather, in software development (and perhaps a few other environments) the combination and intensity of these attributes cause them to become force multipliers.

[18] Evaporating Cloud is a standard TOC Thinking Process, introduced by Goldratt (1990) *What it this thing called Theory of Constraints and how is it implemented*, North River Press. For a summary, see Scheinkopf (2010).

In the real world of software development, this cloud is increasingly 'resolved' in Agile's favor. Development teams feel the benefit of continuous value delivery but are especially drawn to Agile's adaptability, considered to be the primary weakness of waterfall. Agile also provides autonomy and control to the development team. This point cannot be over-emphasized. These things become beautiful *"mermaids" (desirable possessions)* to software developers, who in general are not willing to give them up.

Nevertheless, CCPM's claims to help bring projects in on-time, on-budget and with original scope intact attracts the attention of people in software development as much as they do in other project environments, again especially at the company level. TOC consultants often approach such organizations with deep experience, many great case studies and an impressive resume of results, insisting that CCPM has cracked the code of project management success.

But the interest of the development team turns to concern when they recognize, either from the initial training or in the early stages of trying to use CCPM, that its established techniques are reminiscent of waterfall and in fact create serious practical difficulties for them. They begin to believe that, like waterfall, the established CCPM methodology, despite its benefits, does not possess the adaptability they need, nor the autonomy they desire.

Meanwhile, the growing acceptance of Agile methods in software development makes it advisable for their parent organizations to seriously consider Agile as a solution. The adaptability offered by Agile very soon becomes apparent. This adaptability clearly solves one of the significant needs of the software development organization. Since they can see the logic in the CCPM solution and its ability to deliver on promises, they are left with a similar but very practical conflict as the one between waterfall and Agile, represented in Figure 2 as an Evaporating Cloud. By extension, developers often conclude that although CCPM has impressive merits, the established CCPM methodology appears too like waterfall, especially considering the level of effort required to meet time and budget commitments. Therefore, to them, CCPM is less desirable than Agile.

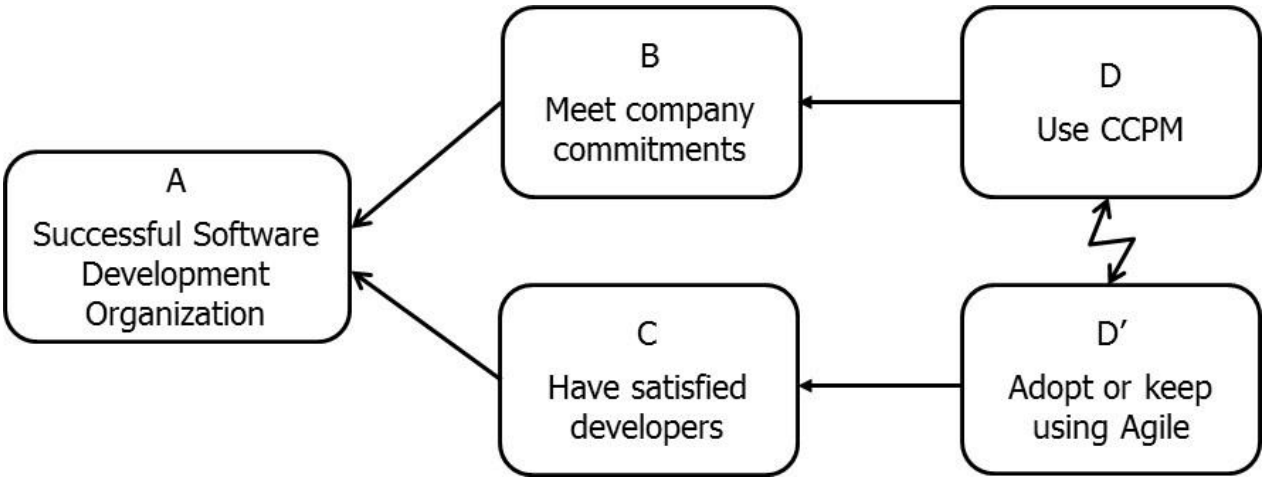

Figure 2 A cloud for software development teams that have been exposed to both Agile and CCPM.

When we begin to dive down into the established techniques of CCPM, we find out why they come to this conclusion. Based on the success of CCPM in other project environments and understanding the cognitive biases noted above, two assumptions have (generally) grown into passive acceptance in the TOC community:

1) Established CCPM techniques are superior to all other techniques in all project environments, including software development.
2) Specifically, every task must be given a numeric time duration estimate, or "Absolute Time Estimate." (ATE)

We claim that neither of these assumptions are valid and this warrants the investigation into alternative CCPM techniques which operate under different assumptions. The resulting investigation has led to a hypothesis which has been tested and the resulting method has shown excellent results in software development.

The proposed method represents an intuitive way to be more 'agile' (adaptable) with some of the techniques used in CCPM and to get greater acceptance and more demonstrable schedule and cost results in software development. It therefore delivers the best of both worlds and leaves on the table no significant need of the software development organization.

This method enhances the power of CCPM in software development by doing time-based buffer management via Relative Size Estimation (RSE) rather than the usual Absolute Time

Estimation (ATE). RSE, also popularly called "T-shirt Sizing[19]," (see Figure 3 below) is the primary injection for the solution. It was popularized from within the world of software development, but has been used in various forms since as early as the 1960s, and perhaps much longer. In the last decade, it has been adopted by advocates of Agile and has been promoted as an Agile best practice.

Relative size, put most simply, means "this thing is about the same size as that thing." In the minds of the engineers, it is determined that a new feature in a software product should require about the same *amount of effort* as a different feature previously developed, even if that feature was completely unrelated. They are of the same *relative size*.

Another way to think about RSE is as a "stable" measurement that does not change due to influence from external forces. The *weight* of two spheres might be 300 Kg and 60 Kg on the earth, but the same spheres would weigh 50 Kg and 10 Kg, respectively, on the moon. However, the relative *size* of the two spheres is stable, not influenced by the external force of gravity. "How long will it take?" in such an environment is like asking "How much does it weigh?" while comparing weight in one part of the solar system to another. The answer can be "I don't know," or, "it depends." But with relative size, the answer does not change – it is stable.

Understanding the primary injection of RSE, the following procedure should be followed to successfully execute the Agile CCPM technique:

Process Step 1. Create a feature list.
Depending upon the environment, create a list of desired features[20]/characteristics/systems or components desired to be developed.
Process Step 2. Prioritize the list by importance.
Process Step 3. Estimate the relative "size" of each feature/characteristic/system/component. This can be done utilizing the established practice of creating *story points.* A very small feature or activity might be classified as one story point. Larger features or activities will be

[19] E.g. XS, S, M, L, XL, etc. More information on T-shirt sizing can be found here: https://dzone.com/articles/agile-estimation-practice .

[20] A "feature" describes a benefit for its users in the form of "user story": "As an A (role), I want to do B (feature), because C (reason)". For example, "As a pipeline manager, I want a multi-project fever chart, because it enables us to see the health of the whole pipeline, and at a glance tells us which projects need our help". In addition, we can obtain both short-term and long-term benefits by using this method to describe features. As a short-term benefit, it helps developers maintain their objectives during the current project. As a long-term benefit, it helps train developers to think more like scientists.

represented by using multiple story points[21]. It is helpful in classification of features to use a modified version of the Fibonacci scale.[22] Estimating relative size by using these numbers allows us to reflect the inherent uncertainty and to avoid underestimation in larger features, since the distance between numbers grows progressively larger.

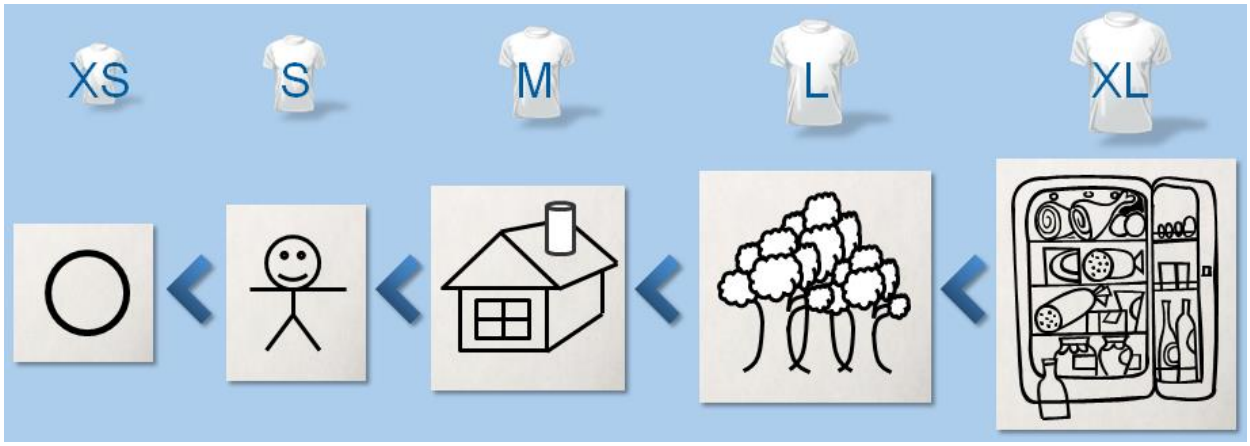

Figure 3 An example of relative size estimation. "Size" here refers not to physical dimensions, but to the level of effort required by the artist in drawing each picture.

As mentioned above, this practice is akin to how one might estimate the relative size of T-shirts (XS, S, M, L, XL, 2XL, etc.) Using the modified scale (see footnote 21), assign a value for each feature/characteristic/system/component desired for inclusion in the project.

Process Step 4. Set project parameters. Several policies need to be set as you create each project. They are illustrated below in Figure 4.

[21] In software development, Planning Poker is often used for this purpose. More information on Planning Poker is available at: http://dzone.com/articles/introduction-planning-poker. A good article on story points can be found here: https://myagilemind.wordpress.com/2011/10/18/story-points-vs-development-hours.

[22] http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/fib.html. The standard Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, etc., where each successive number in the sequence is the sum of the previous two. However, for the purposes of estimating relative size in the proposed method, the sequence may be modified by rounding to more practical values: 0, 1, 2, 3, 5, 8, 13, 20, 40, 60, 100, etc. These numbers now help us to measure relative size, or the number story points for each feature.

4a. Set the project period (duration) in days. Identify, prioritize, and group target features by relative size (in story points). Sum the number of story points.
4b. Determine and set the planned initial velocity, or number of story points to accomplish per day.
4c. Divide the number of story points by the initial velocity. The resulting number is the project size in days.
4d. Once the total duration of the project is known, select a buffer coefficient. A good starting point is generally 33% of the project period[23]. Figure 4 below shows how buffer coefficient selection would look prior to Critical Chain scheduling.

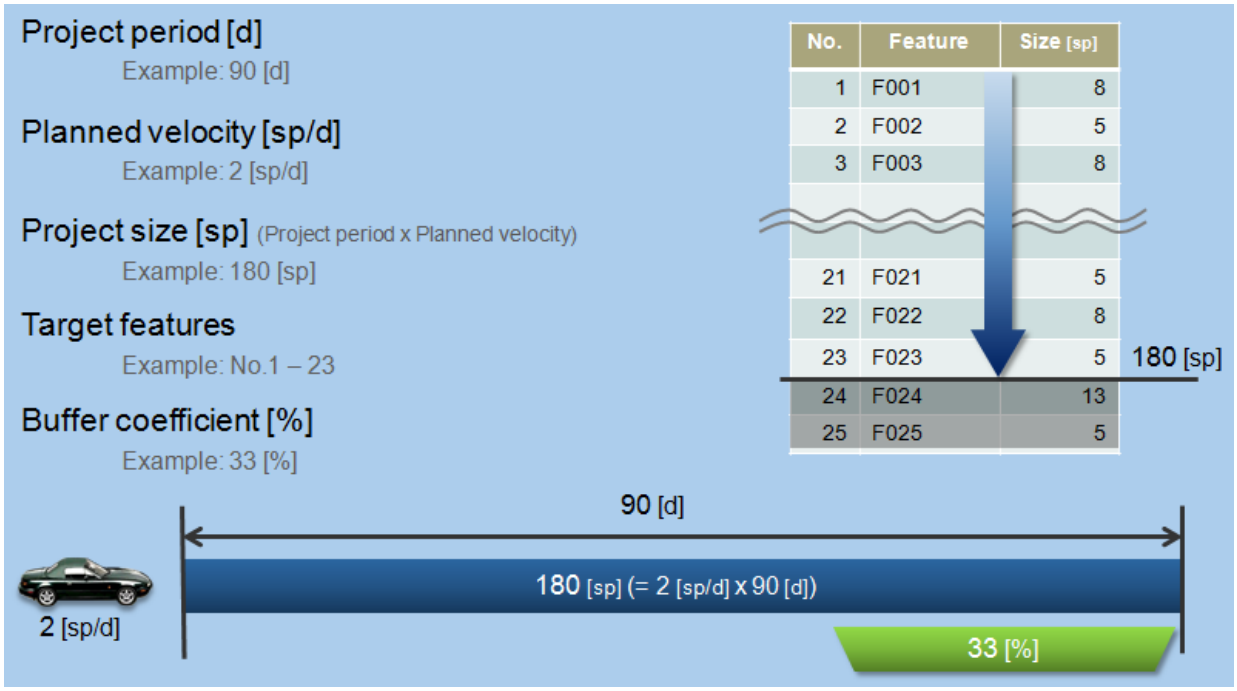

Figure 4 Setting project parameters. A selected buffer coefficient is shown, which will be applied during Critical Chain Scheduling.

[23] This produces a different result from a common practice in CCPM, where 50% of task time is removed from each task as safety, and half the removed and aggregated safety is placed at the end of the path as a buffer. In such a case, the resulting project is 75% of its non-CCPM scheduled size. But since features are not technically tasks, in Agile CCPM we want the duration of the Critical Chain plan to be the same as in the original project. Therefore, a project which is set to 90 days prior to Critical Chain Scheduling will still be 90 days after Critical Chain Scheduling. Rather than task time being 50% of its pre-CCPM value, in Agile CCPM task time is 67% of its pre-CCPM value. Arriving at this configuration may require different techniques between unique CCPM software products.

Process Step 5. Create a Critical Chain schedule.
Very simple schedules are adequate for this method – for single projects sometimes even a single path and the equivalent of one (or a very few) "tasks[24]," and a project buffer are sufficient. (See figure 5).

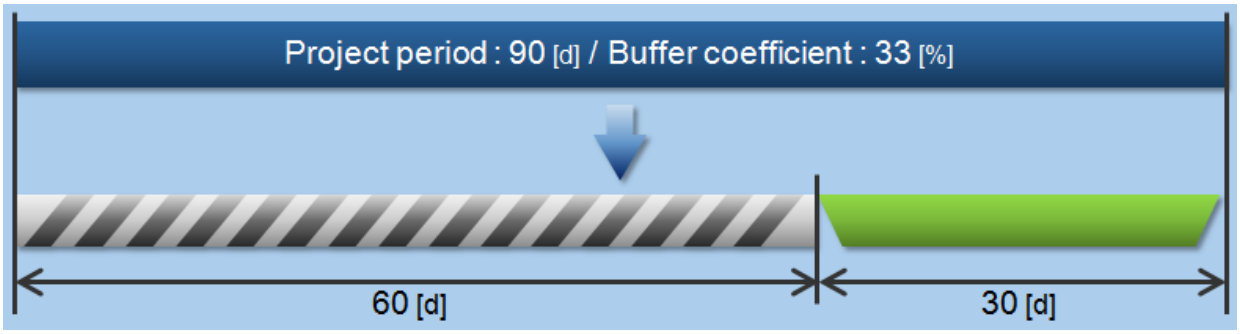

Figure 5 The same project as shown in Figure 4 after Critical Chain Scheduling. A very simple schedule is sufficient for single project CCPM in software development

More common in the project world are multi-project organizations, yet still the network can be very simple – a minimal number of tasks. An integration phase can be identified, and projects can be staggered based on the Virtual Drum phases of projects in the pipeline.
5a. Example: Estimate a duration for the developer test phase based on experience and the relative size of the project. This is a prime candidate for the integration phase/Virtual Drum.
5b. Determine the duration of the QA task (often a fixed duration for all projects set by policy). Sometimes this task is combined with the developer test phase in integration. (See Figure 6)

Figure 6 Critical chain with integration phase for Virtual Drum staggering

Some projects may require or benefit from one or more feeding paths, for example when different skill sets can be assigned to features and work can be done in parallel. Multiple paths may also be used when some features are being outsourced. In addition, if there are significant

[24] i.e. groups of features, or a phase of the project

differences in skill levels among resources, slower resources could be assigned to a feeding path. All paths should be managed per their own velocity. With these multiple paths, there will also be feeding buffers, as in standard CCPM. (See Figure 7)

Figure 7 Project network showing feeding chain and Virtual Drum.

5c. Designate a "Scope Buffer." Scope creep, which has a potential for existence in all types of projects, can be pervasive and extreme in software development. Not only are projects subject to customer input and change, sometimes the software development environment itself generates new ideas and improvements which add or change content.
Whereas other project environments might be categorized as in the realm of "known unknowns"[25], software development can often exist in the realm of "unknown unknowns"[26]. Standard CCPM contributes powerfully to scope preservation in most environments by optimizing project execution. However, in software development the final scope of the project is very often unknown. It is not necessarily practical or desirable to preserve the original scope. As a new feature is developed, often an important addition to the scope of the feature is discovered. In execution, and/or in the full kitting phase, stakeholders may change their minds. This can result in feature inflation or changing the priority of remaining features.

Experience shows that in software development scope inflation can be as much as 30%. This is out of the range of normal buffer management, therefore creating the need for a scope buffer. The scope buffer is a virtual buffer. If the crucial features are properly prioritized and maintained, a scope buffer allows us to defer or replace less important features – with consensus among stakeholders.

In execution, when we see a situation where the project buffer is about to be exhausted, decisions can be made on which features can safely be removed from the list. To facilitate this,

[25] A supplier might be late, a key piece of equipment may break down, a subcontractor may not be available, etc.
[26] Alternatively, in software development, the ratio of unknowns may be relatively higher.

the last 15% of the initial planned scope should be designated as the scope buffer[27]. The scope buffer is set in the planning phase, once and only once, and will be put in motion as necessary. In other words, this means also that developers promise stakeholders to defend at least 85% of the original contents "to the death".

Priority	Feature	Size (sp)	Cumulative	Pct.
1	F001	8	8	7.0%
2	F002	5	13	11.4%
3	F003	8	21	18.4%
4	F004	13	34	29.8%
5	F005	5	39	34.2%
6	F006	5	44	38.6%
7	F007	8	52	45.6%
8	F008	5	57	50.0%
9	F009	5	62	54.4%
10	F010	13	75	65.8%
11	F011	8	83	72.8%
12	F012	5	88	77.2%
13	F013	8	96	84.2%
14	F014	5	101	88.6%
15	F015	13	114	100.0%

Figure 8 Feature list showing last 15% as scope buffer. Note that the 15% mark is initially within the planned time for Feature #13

5d. Stagger projects as required.

Staggering projects should be done in the same manner as standard CCPM[28], keeping the following basic principles in mind:

1) Top management prioritization of projects (per Projects S&T 5.111.1)[29]
2) Proper CCPM network and pipeline planning (including employment of a Virtual Drum and capacity buffers)

[27] If the ratio of scope inflation is recognized as being up to 30%, and the half of the new features would have higher priority than features in the original scope, 15% of the original features can be replaced by emerging requirements.

[28] If resources are shared and Virtual Drum staggering is used consistently across all projects, it is permissible to mix Standard CCPM and Agile CCPM projects in a pipeline.

[29] Projects Strategy & Tactics Tree, Goldratt, October, 2010

3) Maintaining low WIP levels via determination of a maximum number of projects to be allowed concurrently in the Virtual Drum phase

As with standard CCPM, project completion dates are ideally committed based on where a project's completion date falls in the stagger. Therefore, lead times and completion dates are determined by collaboration between sales and the pipeline manager. Most CCPM software products allow project delivery dates to be calculated by utilizing the "what if" capabilities of the software in accordance with the above.
As with standard CCPM, if the resulting completion date is unacceptable, some top management decisions may be required regarding scope and/or prioritization position in the stagger.

5e. Start projects as per the pipeline stagger and perform velocity-based buffer management. The start of projects should also be in accordance with basic CCPM and S&T tree principles:

1) Release of project "legs" (S&T 5.111.4).[30]
2) Release of projects is subject to the guidelines for full kitting (S&T 4.11.2[31] and below).

To perform velocity-based buffer management, or VBBM, the following procedure should be followed:

1) Initial Buffer Management Procedure – As is the case with standard CCPM, remaining task duration should be estimated in a timely manner, such as at the end of the working day. Task managers should report remaining duration in days (also as with standard CCPM). We believe this can be done relatively easily once features are in execution. However, if it seems that there are difficulties in estimating remaining duration directly (by ATE), estimation can be accomplished by dividing the number of remaining story points (remember a task can include several features) by the planned velocity.[32]

   If multiple portions of the task are incomplete at the end of a working day, the task manager should report remaining duration based on the portion of the task with the highest remaining duration. This procedure will be maintained until the duration of the

[30] Ibid.
[31] Ibid.
[32] Note however, that difficulty in absolute time estimation at this point would be a sign that the definition of the feature is too vague. Another sign that the task definition is too vague is that task level full-kitting is difficult to identify.

tasks for remaining features can be re-calculated by using actual velocity instead of initial planned velocity[33].

Initial buffer color is calculated measuring buffer consumed relative to the planned Critical Chain duration. This represents no change from the existing reporting method incorporated by Critical Chain software.

2) Secondary Buffer Management Procedure after switching to actual velocity - When the current feature and upcoming features are represented as individual tasks (i.e. [Task 1 = current feature] -> [Task 2 = upcoming features]), then 1) report remaining duration (RD) for the current feature in the same way as with standard CCPM, and 2) adjust the duration of upcoming features by dividing the remaining story points by actual velocity (only the 2nd step is different from the standard CCPM) (see Figures 9, 10).

   When both the current and upcoming features are represented by a single task bar, report the value obtained by the following equation as the remaining project duration:

   Remaining project duration = RD + ED

   where [d] = days, [sp] = story points, and

   RD [d]: (The highest) remaining duration of the current feature(s)

   ED [d]: Estimated duration of remaining features [SPr / V * (1 – Cb)]

   SPr [sp]: Story points of remaining features[34]

   V (Velocity) [sp/d] = SPc / Te

   SPc [sp]: Story points of completed features

   Te [d]: Elapsed time [d][35]

   Cb [%]: Buffer coefficient

[33] Project velocity, in the early stage of a project, can be unstable. Velocity is calculated from the accumulated value of story points and elapsed time, which means that as each data point is recorded, the ratio of fluctuation decreases gradually. When you are confident that actual velocity has stabilized (e.g. after 10 days), change the procedure for reporting your remaining duration estimates from calculating them against initial planned velocity to calculating them against the average actual velocity of completed features.

[34] Even if a portion of the current feature is completed, e.g. 4 SPs out of 8 SPs, don't count these as completed story points. Experience shows that trying to pinpoint the precise number of completed story points is out of proportion to the benefit. ("It is better to be approximately right than precisely wrong")

[35] To simplify the explanation in this paper, elapsed time is shown in days. In practice, elapsed time should be counted hourly and overtime work should be considered to obtain a more reliable buffer status.

The value is entered as the remaining duration in days for the task representing the project (single path), and this can be done in standard CCPM software (see Figure 11). An easy way to visualize the concept is to think of a project as analogous to going on a long drive:

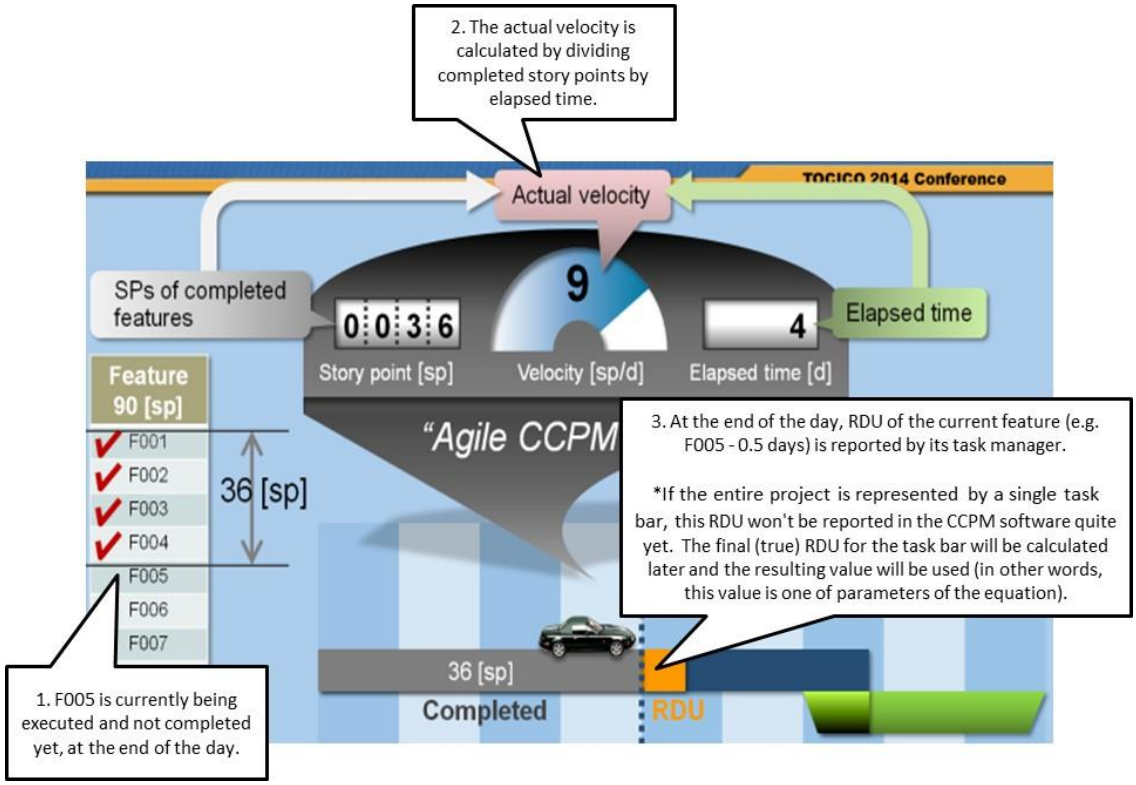

Figure 9 How to calculate actual velocity

3) As with standard CCPM, analyze the remaining chains and take actions as necessary to recover red buffers to yellow or green

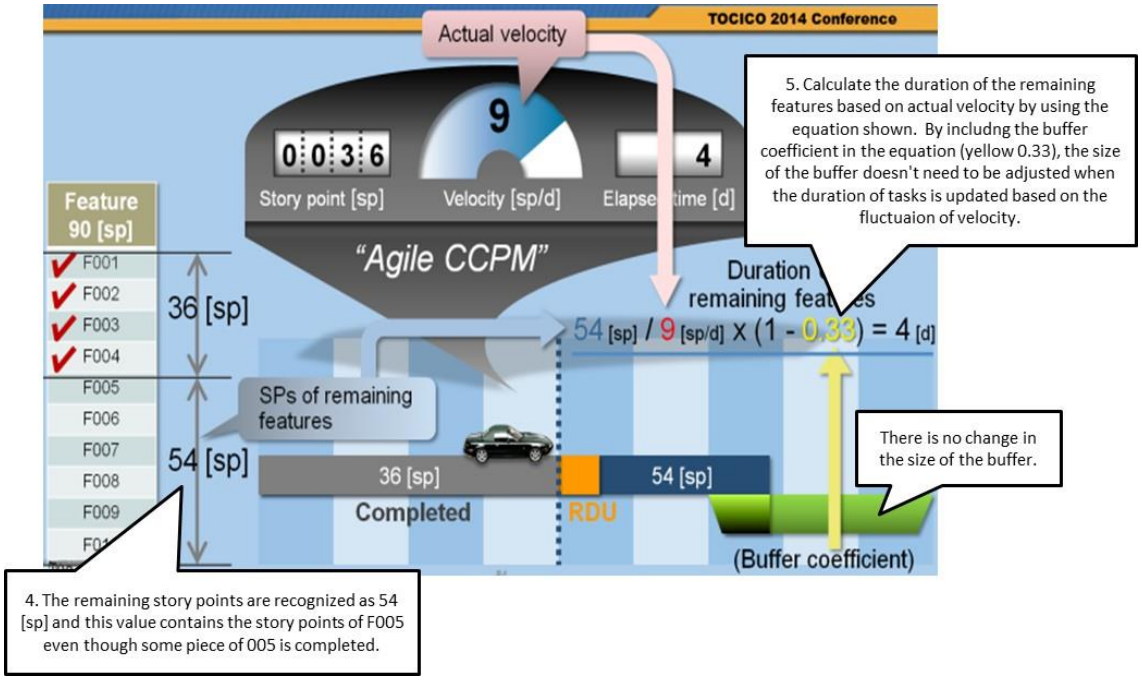

Figure 10 Calculating the duration of remaining features using actual velocity

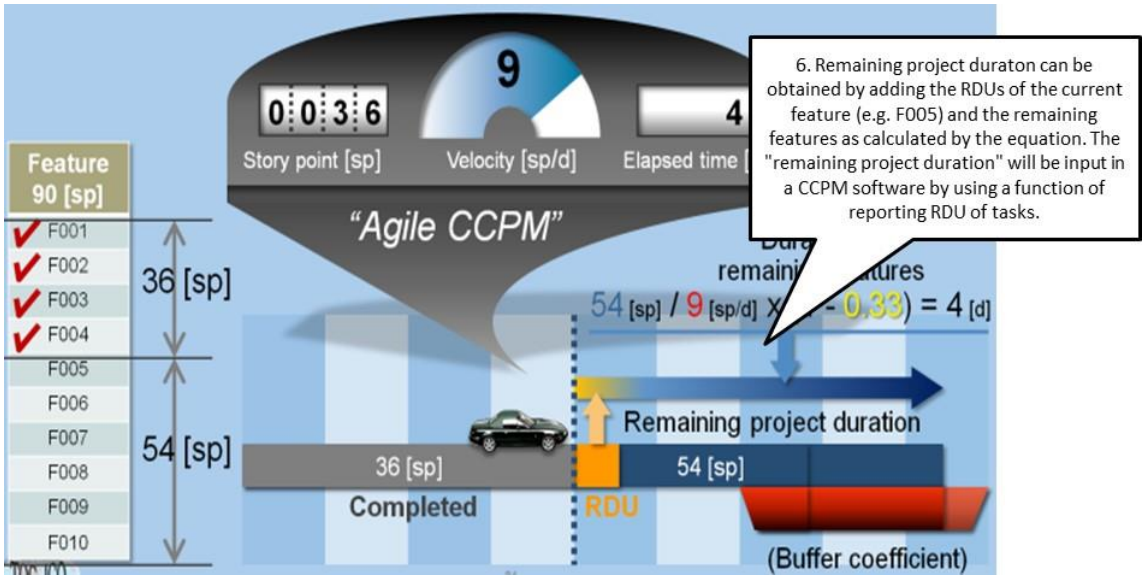

Figure 11 Calculating the remaining project duration. In this example, the buffer is shown notionally as changing to red per the new remaining duration.

Brief case study of "Agile CCPM" as applied in software development

This case study reflects the experience of Being Company Ltd. in Japan, the employer of one of the authors and main developer of Agile CCPM, Koichi Ujigawa.

Being Company Ltd. first began using CCPM for its project management in early 2005. Although some improvement was demonstrated, many difficulties remained, and the expected 95% on-time project completion rate was far from being obtained. Even though Being was using standard CCPM, DDP was less than 30%.

Amid these conditions, Being was faced with scheduling a major upgrade of their software. Because most of the source code had to be developed from scratch and historical performance data for such an upgrade was not available, the Absolute Time Estimation of tasks was extremely difficult, and thus creating a reliable CCPM schedule seemed nothing short of impossible.

Therefore, the development team had serious reservations about managing the project with CCPM, even though they still firmly believed in the power of the Theory of Constraints. The only thing they knew conclusively was that the project was much larger than anything they had attempted in the past, and the company was afraid that the project would badly miss its delivery schedule unless a solution was found and a change was made in the managing of the project.

Thus, while remaining committed to the principles of CCPM, in mid-2009 Being began to introduce ideas used in Agile methodology (e.g. Relative Size Estimation and "sprints", although it was subsequently learned that sprints are not necessary for Agile CCPM – see below), for its CCPM implementation.

Although some projects still missed their due dates in the early stages of Being's effort, DDP was improved gradually, and the resulting organic method known as Agile CCPM was fully developed and finally presented at the 2012 TOCICO International Conference.[36]

DDP had reached over 90% by the end of 2011, and with the perfecting of the Agile CCPM method, has been maintained at almost 100% in the subsequent years. A recent snapshot of Being's project performance, managed under "Agile CCPM", can be seen in Figure 11 below.

[36] Ujigawa, Koichi, Goldratt Foundation semi-finalist: "A New Buffer Management Approach for CCPM" 2012 TOCICO International Conference, http://c.ymcdn.com/sites/www.tocico.org/resource/resmgr/2012_foundation_pdfs/ujigawa,_koichi_final_a_new_.pdf

Name		% PB Consumed (PB Finish)	% CC Completed (CC Start)
Project A		39.7	100.0
Project B		12.2	100.0
Project C		39.7	100.0
Project D		99.8	100.0
Project E		63.0	100.0
Project F		90.8	100.0
Project G		85.1	100.0
Project H		69.7	100.0
Project I		69.9	100.0

Figure 12 Being Company Ltd's own completed software development projects, managed as per the process described in this paper. Certain business sensitive information has been obscured.

Conclusion

As with standard CCPM, it is important to plan for and assign optimal resources to tasks and projects as described in the Projects S&T tree (5.111.2)[37], and to ensure full kits (4.11.3 and below). Due to the optimal assignment of resources, full kitting, use of a scope buffer, and a proper stagger (Projects S&T 5.113.3), the project period is sufficient to yield on-time or early project completion.

"Agile" CCPM, a CCPM method for software development, delivers the adaptability provided by Agile, as well as the cost, delivery and scope performance afforded by CCPM. The result is greater than the sum of its parts, and much greater than the result of any existing methodology, including traditional CCPM or Agile alone. This allows a large new market sector of projects to be executed under the CCPM umbrella, further helping TOC to become the main way in business, and is therefore deserving for inclusion in the TOC Body of Knowledge.

[37] Projects Strategy & Tactics Tree, Goldratt, October, 2010

References

Critical Chain

Goldratt, Eliyahu M. 1997 *Critical Chain.* Great Barrington, MA, North River Press.

Goldratt, Eliyahu M. October, 2010 *Projects Strategy & Tactics Tree,* detailed in *The Critical Chain Implementation Handbook* by David Updegrove (below), and available from that author in Harmony software file format by request at davupde@msn.com.

Updegrove, David B. 2014 *The Critical Chain Implementation Handbook*. Createspace Publishing.

Goldratt's Change Matrix

Scheinkopf, Lisa, 2010 *Thinking Processes Including S&T Trees*, Chapter 25, *Theory of Constraints Handbook*, edited by James F. Cox III and John G. Schleier, Jr., McGraw Hill.

Fergusson, Lisa, *Applications of Strategy and Tactics Trees in Organizations*, Chapter 34, *Theory of Constraints Handbook*, edited by James F. Cox III and John G. Schleier, Jr., McGraw Hill.

Barnard, Alan, 2016 *The Change Matrix Cloud Process,* Draft White Paper, TOCICO White Paper Series, TOCICO.

"Agile" CCPM

Ujigawa, Koichi. *A new buffer management approach for CCPM.* TOCICO International Conference 2012, Chicago, Illinois, USA http://www.tocico.org

Ujigawa, Koichi. *A new buffer management approach for CCPM.* Encore presentation. TOCICO International Conference 2013, Bad Nauheim, Germany http://www.tocico.org

Ujigawa, Koichi. *Agile CCPM*. TOCICO International Conference 2014, Washington, D.C., USA http://www.tocico.org