UNIVERSIDAD ANDINA SIMÓN BOLÍVAR
SEDE ECUADOR

ÁREA DE GESTIÓN
PROGRAMA DE MAESTRÍA EN DIRECCIÓN DE EMPRESAS

LA TEORÍA DE RESTRICCIONES APLICADA AL DESARROLLO DE SOFTWARE

NICANOR PALACIOS ALVAREZ

Marzo, 2010

Al presentar esta tesis como uno de los requisitos previos para la obtención del grado de magíster de la Universidad Andina Simón Bolívar, autorizo al centro de información o a la biblioteca de la universidad para que haga de esta tesis un documento disponible para su lectura según las normas de la universidad.

Estoy de acuerdo en que se realice cualquier copia de esta tesis dentro de las regulaciones de la universidad, siempre y cuando esta reproducción no suponga una ganancia económica potencial.

Sin perjuicio de ejercer mi derecho de autor, autorizo a la Universidad Andina Simón Bolívar la publicación de esta tesis, o de parte de ella, por una sola vez dentro de los treinta meses después de su aprobación.

Nicanor Palacios Álvarez

Marzo del 2010

UNIVERSIDAD ANDINA SIMÓN BOLÍVAR
SEDE ECUADOR

ÁREA DE GESTIÓN
PROGRAMA DE MAESTRÍA EN DIRECCIÓN DE EMPRESAS

LA TEORÍA DE RESTRICCIONES APLICADA AL DESARROLLO DE SOFTWARE

NICANOR PALACIOS ALVAREZ

Quito, marzo de 2010

Tutor: Ing. José Miguel Fernández

Resumen

Esta tesis busca definir una alternativa para administrar los proyectos de desarrollo de software basándose en la Teoría de Restricciones del Dr. Eliyahu M. Goldratt y siguiendo los principios de la metodología Ágil Scrum, con el fin de lograr proyectos exitosos que satisfagan a los clientes y sean rentables para la compañía ejecutora.

En el capítulo 1 se presenta el marco lógico de la Teoría de Restricciones y los 5 pasos propuestos para implementar un proceso de mejora continua identificado el elemento que limita al sistema y le impide generar los resultados financieros deseados.

Se continúa con el análisis de las particularidades de los proyectos de desarrollo de software, las bases que sustentan a las metodologías de Desarrollo Ágil y se estudia en detalle a Scrum.

En el capítulo 3 se conjugan las reglas del marco de trabajo Scrum con los lineamientos de la TOC para obtener una guía práctica para administrar los proyectos de desarrollo de software. Se evidencia que el proceso de desarrollo es un sistema con una restricción que lo limita y que se pueden aplicar los 5 pasos propuestos por la TOC para optimizarlo. Además, los argumentos lógicos de la Cadena Crítica, aplicación de la TOC para administrar proyectos, permite asignar amortiguadores en los lugares adecuados de la ruta del proyecto haciendo que el Scrum Master se enfoque en los desfases causados por los inevitables imprevistos.

En el capítulo 4 se analizan las convenciones de la contabilidad de Throughput para definir indicadores de operación y de resultados que nos permitan monitorear y controlar los avances de los proyectos, determinar su éxito y poder comparar las continuas mejoras conseguidas con la aplicación de la TOC y Scrum.

En el último capítulo el lector podrá encontrar la aplicación de la teoría analizada en un proyecto real que ha sido seleccionado para ilustrar de mejor manera el marco de trabajo propuesto.

A mi esposa Andrea, por su paciencia y apoyo incondicional,
a mis padres y hermana por su inmenso cariño y soporte.

Agradecimientos

Deseo agradecer a mi tutor de esta tesis, Ing. José Miguel Fernández, por su valiosa guía, seguimiento y contribuciones realizadas durante todo el proceso de elaboración de la misma. También agradezco a la UASB, sus profesores y personal administrativo por el esfuerzo que realizan diariamente para mejorar el nivel de los profesionales del país.

Un agradecimiento muy especial a todos mis compañeros por haber hecho que los dos años de estudios se hayan convertido en una experiencia memorable.

ÍNDICE

INTRODUCCIÓN

Esta tesis de postgrado tiene como objetivo presentar una alternativa para administrar los proyectos de desarrollo del Software basándose en la ideas de la Teoría de Restricciones creada por el Doctor Eliyahu M. Goldratt y los principios de las metodologías de Desarrollo Ágil.

El proceso de desarrollo de software es altamente complejo e involucra a cinco actores indispensables: los requerimientos del sistema, la tecnología utilizada, la metodología empleada para la gestión y control del proyecto; los desarrolladores y analistas; y, los usuarios finales. Cada uno de estos actores incorpora un grado de complejidad, que en conjunto pueden crear un nivel de incertidumbre que se vuelve inmanejable.

Un primer problema muy frecuente es que desde el inicio del proyecto, los requerimientos no son o no pueden ser definidos con exactitud, y éstos son cambiados a lo largo del mismo; lo cual ocasiona que el diseño inicial del software a desarrollarse termine siendo muy diferente a la versión final entregada.

En segundo lugar, la tecnología usada para el desarrollo de software trata de resolver los problemas y cubrir las necesidades de los usuarios, pero nunca su adaptación es simple. Además la tecnología evoluciona con tanta rapidez que resulta difícil que tanto los desarrolladores, analistas y usuarios finales logren usarla óptimamente.

El tercer elemento a considerar es la metodología empleada para administrar el proyecto de desarrollo y su control. En muchas ocasiones los proyectos son ejecutados sin enmarcarse en alguna metodología de desarrollo y por tanto se pierde toda posibilidad de realizar una planificación y control adecuados incrementando drásticamente las posibilidades de fracaso del proyecto.

El problema final radica en conciliar la visión propuesta por el equipo de desarrollo y las expectativas de los usuarios finales. Las divergencias rara vez son detectadas en etapas tempranas de los proyectos y sólo se evidencian luego de realizar grandes avances en el desarrollo, por lo que la implementación de cambios requiere de mucho esfuerzo y recursos.

En busca de una solución al problema expuesto y debido a que la Teoría de Restricciones ha tenido gran aceptación a nivel mundial y ha sido ampliamente utilizada para la administración de proyectos, esta teoría se considera como una herramienta que al ser implementada eficientemente mejorará los procesos en el desarrollo de software generando grandes beneficios para esta industria.

Al implementar el proceso lógico de pensamiento de la Teoría de Restricciones en los proyectos de software se podrá mitigar la complejidad de los mismos. Además, ya que la norma en un proyecto de desarrollo es su urgencia y criticidad se ha seleccionado analizar las metodologías Agiles de Desarrollo y principalmente la metodología Scrum para que sea el marco de trabajo en los proyectos. Por último, se van a definir indicadores útiles basándose en la contabilidad de Throughput[1], propuesta por la Teoría de Restricciones, se presentarán mecanismos de medición y control para las etapas del desarrollo de software.

[1] Thomas Corbett, en *Throughput Accounting*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1998, describe a la contabilidad de Throughput como una metodología alterna a la Contabilidad de Costos para la toma de decisiones en empresas, se basa en los principios de la Teoría de Restricciones de Eliyahu M. Goldratt. William Dettmer, en *Goldratt´s Theory of Constraints*, Estados Unidos de América, Editorial ASQC Quality Press, 1997, define a la palabra Thoughput como “la tasa a la cual todo el sistema genera dinero a través de las ventas”, es decir la cantidad de dinero proveniente del sistema.

CAPITULO 1.
INTRODUCCIÓN A LA TEORÍA DE RESTRICCIONES (TOC)

1.1. Los sistemas como cadenas y su restricción

La Teoría de Restricciones fue creada por el Doctor Eliyahu M. Goldratt, quien ha escrito varios libros en donde ha formulado, analizado y profundizado los conceptos y las herramientas de esta Teoría. Esta filosofía de administración ha sido usada por numerosas empresas pertenecientes a varias industrias a nivel mundial como son: manufactura, construcción, administración de proyectos, entre otras, generando beneficios en la optimización de sus procesos.[2]

La Teoría de Restricciones, TOC (generalmente conocida por sus siglas en inglés, Theory of Constrains) es una metodología científica y de pensamiento lógico, creada originalmente para el ámbito industrial de producción. Esta teoría permite a las empresas conseguir mejoras sustanciales en su desempeño enfocándose principalmente en dos cosas: la meta a la que se quiere llegar y la restricción que impide lograr la meta.

La TOC en ambientes de producción industrial busca identificar los cuellos de botella en el proceso de producción. Esto se realiza ya que se asume que una línea de producción es tan rápida como el proceso más lento en la cadena productiva. La capacidad del proceso más lento determina la restricción de todo el sistema.

El Dr. Goldratt define como sistema a una red de componentes interdependientes que trabajan conjuntamente para alcanzar una meta[3], esta definición general se ajusta perfectamente a la definición de un sistema informático: un conjunto o disposición de elementos que están organizados para cumplir una meta predefinida al procesar

[2] En el estudio independiente realizado por los profesores Steven J. Balderstone y Victoria J. Mabin de la universidad de Wellington en New Zealand se concluye que en promedio las 100 empresas analizadas que han implementado la Teoría de Restricciones han incrementado sus utilidades en 68%.
[3] Eliyahu M. Goldratt, y Fox, Robert E, *La Carrera*, México, Ediciones Castillo, 1999. pág. 17.

información.[4] Cada componente se encuentra relacionado con los demás, lo que obliga a tomar decisiones siempre tomando en cuenta su impacto en el sistema como un todo. Por esto es indispensable entender cómo los elementos interactúan entre si.

Muchas veces, se acepta la idea errada que la suma de los esfuerzos individuales de los componentes o departamentos de una empresa da como resultado el rendimiento global de la compañía. Esta falacia asume que cada componente del sistema trabaja de una manera aislada e independiente. Por esta razón se han buscado obtener óptimos locales en cada departamento perjudicando el rendimiento global de la compañía y alejándola de sus objetivos.

A diferencia de la premisa anterior, el Dr. Goldratt enfatiza que solo hay un recurso que restringe la capacidad de un sistema en un momento dado, que es entonces considerado el eslabón más débil de la cadena. En ambientes de producción la restricción puede ser una maquinaria o centro de trabajo en particular, en un ambiente de trabajo intelectual será una persona o un grupo con las habilidades específicas para prestar el servicio. Todos los demás procesos del sistema, en ese intervalo de tiempo, son recursos no restricciones y no determinan la capacidad del sistema.

Si se desea incrementar la capacidad del sistema, y se conoce que existe un único recurso restricción, resulta obvio que se debería centrar todos los esfuerzos en este recurso que está limitando la capacidad global. Resultaría infructuoso mejorar cualquiera de los procesos no restricción ya que la capacidad del sistema no incrementaría. No hay necesidad de enfocarnos en el resto de recursos ya que de antemano sabemos que tienen capacidad ociosa y pueden afrontar las fluctuaciones de trabajo.

En la figura 1 se presenta el proceso de desarrollo de software y para cada etapa se ha colocado el número de unidades de código que puede procesar en una semana.

---

[4] Roger S. Pressman, *Ingeniería del Software*, un enfoque práctico, México, McGraw-Hill, 2006, pág. 134.

Se puede ver claramente que la actividad de pruebas del sistema tiene una capacidad de 30 unidades, la más baja de la cadena, por lo que es el recurso restricción.

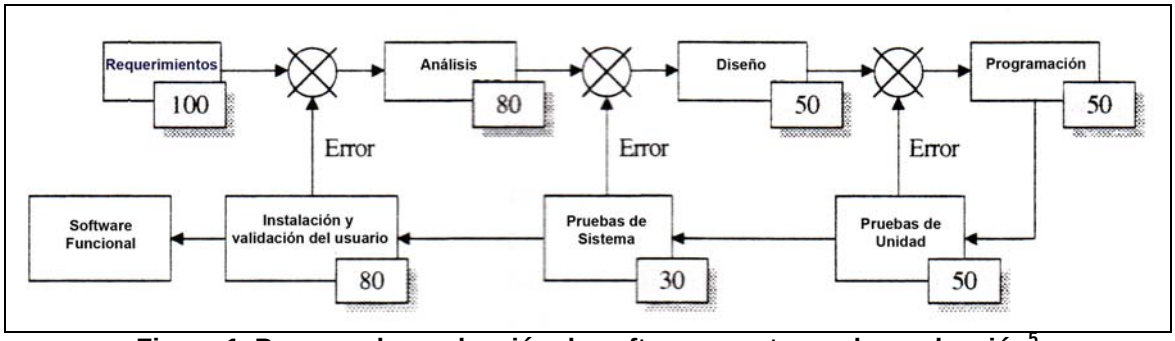

Figura 1. Proceso de producción de software con tasas de producción[5]

Todo sistema busca alcanzar resultados concretos, a estos resultados se los conoce como la meta del sistema. Como se mencionó anteriormente, es la restricción del sistema lo que impide cumplir los objetivos. Al incrementar la capacidad de recurso restricción, considerado como el eslabón más débil de la cadena en ese momento, se puede conseguir que ese recurso deje de ser la restricción del sistema y se habrá dado un paso para lograr la meta.

1.2. La meta del sistema

La teoría de Restricciones busca mejorar la productividad de un sistema al identificar su restricción y optimizarla para alcanzar la meta deseada. Se debe saber con claridad cuál es esa meta del sistema, para enfocar los esfuerzos y poder medir y controlar si se está llegando a ella.

Es muy frecuente escuchar que la meta de una compañía es brindar productos o servicios de calidad; otras veces se menciona que la meta es satisfacer las necesidades de los clientes; alcanzar ventajas competitivas o participación en el mercado. Todos esos objetivos pueden ser bonitos enunciados y muchas veces se pueden convertir en

[5] David Anderson, *Agile management for software engineering*, Estados Unidos de América, Editorial Prentice Hall PTR, 2004, pág. 30.

caminos o puentes para alcanzar la verdadera meta de una compañía. Hay que ser muy franco y reconocer que la meta es una sola para las empresas con fines de lucro: ganar dinero en el presente y también en el futuro.[6]

Una vez que se ha definido cuál es la meta del sistema se debe analizar si las decisiones tomadas diariamente están encaminando a la empresa hacia ella. Son de conocimiento general los indicadores de resultados para saber si estamos ganando dinero o no. El primer indicador de la cantidad de dinero ganado son las utilidades generadas por la compañía (UN); pero este solo indicador no es suficiente ya que es necesario conocer el monto de la inversión realizada por los accionistas para poder relacionar con las utilidades y confirmar si el monto de las utilidades es satisfactorio comparando con la inversión, este indicador es conocido como retorno sobre la inversión (ROI).

Los dos indicadores mencionados permiten saber la cantidad de dinero ganado y si se está llegando a la meta. Pero no se debe olvidar el indicador fundamental de supervivencia: el flujo de efectivo. Una compañía puede tener buenos niveles de utilidades y de retorno sobre la inversión y sin embargo estar con amenazas de quiebra por no contar con buenos niveles de liquidez. Por esto es vital controlar el flujo de efectivo junto con los otros indicadores.

Los tres indicadores son suficientes para medir las ganancias de dinero generadas por una compañía pero resultan inadecuados para medir el impacto de las acciones y decisiones diarias en las operaciones de la compañía. Se vuelve imperante contar con indicadores que nos ayuden a tener un puente entre las operaciones de la compañía y los indicadores de resultados globales.

Generalmente las compañías se han apoyado en la contabilidad de costos o medidas de productividad para tomar las decisiones en las operaciones diarias de la

---

[6] Eliyahu M. Goldratt, y Fox, Robert E, La Carrera, México, Ediciones Castillo, 1999. pág. 22.

compañía: tamaño de lote a producir, niveles de inventario, mejoramiento de equipos, márgenes de utilidad de productos y su rentabilidad en el mercado. La TOC es muy crítica sobre la búsqueda de óptimos locales, la contabilidad de costos y los principios que la gobierna. Se analizará este tema a profundidad en el capítulo 4 de este trabajo, en este apartado mencionaremos brevemente la propuesta de indicadores desarrollados para la TOC.

La TOC cuenta con tres indicadores operativos globales que sirven de puente hacia los indicadores de resultados, los cuales son:

Throughput (T)[7]: Es la velocidad a la que el sistema genera dinero a través de las ventas. Es muy importante tomar en cuenta que sólo se genera Throughput cuando existe la venta y el pago; y no es suficiente haber producido el producto.

Inventario – Inversión (I): Todo el dinero que el sistema invierte en la adquisición de cosas que pretende vender. Este concepto de inventario difiere del concepto tradicional de la contabilidad ya que no incorpora el valor agregado de la mano de obra ni los gastos generales de fabricación. El Inventario es transformado en el sistema y se convierte en Throughput a través de las ventas.

Gastos de Operación (OE): Se considera a todo el dinero que el sistema gasta en trasformar el Inventario en Throughput. Esta definición de gasto de operación incluye, además de los costos de mano de obra directa y costos indirectos de fabricación, también a los gastos administrativos.

Para resumir podemos decir: el Inventario es toda la inversión en materia prima necesaria para el sistema; el Throughput es el dinero generado por el sistema; y el Gasto Operativo es lo que necesitamos gastar para que el sistema funcione.

[7] El Lic. Nicholas A. Gibler, traductor de *La Carrera,* sugiere respectar el término Thoughput como término técnico de la teoría de restricciones y no usar su traducción al español. En el contexto de la teoría de restricciones Throughput significa “generación de dinero”. En este mismo sentido en este estudio se utilizarán las siglas en inglés T-I-OE para Throughput, Inventory y Operating Expenses.

A continuación se presenta las fórmulas para calcular los indicadores de resultados en base a los indicadores operativos:

UN = T – OE ROI = U / I U: Utilidad T: Throughput OE: Gastos de Operación I: Inversión ROI: Retorno sobre la inversión

Tabla 1. Indicadores de resultados[8]

Los tres indicadores globales de operación tienen un impacto evidente sobre los indicadores de resultados financieros definidos. Por esta razón se convierten en el puente idóneo entre las decisiones de las operaciones y la meta final del sistema.

Cuando se incrementa el Throughput sin afectar los gastos de operación e inventario se influye directamente de forma positiva sobre la utilidad, retorno sobre la inversión y flujo de efectivo. Se obtiene el mismo impacto positivo si se reducen los gastos de operación sin disminuir el Throughput o los inventarios. Al disminuir los inventarios, el monto invertido, también se afecta positivamente el retorno sobre la inversión y flujo de efectivo.

Se puede concluir que las acciones correctas en las operaciones son las que causan por lo menos uno de los siguientes cambios (Δ) sin perjudicar a los otros:

- Incrementar el Throughput.
- Reducir la inversión e inventarios.
- Reducir los costos de operación.

Al unir los indicadores de operación con los indicadores globales de rendimiento se obtiene una clara guía que sirve de base para tomar decisiones que nos acerquen a la

[8] Domenico Lepore y Oded Cohen, *Deming and Goldratt The Theory of Constraints and The System of Profound Knowledge*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1999, pág. 26.

meta planteada. Las siguientes expresiones evalúan el impacto de las decisiones tomadas y confirman su viabilidad:

$\Delta UN = \Delta T - \Delta OE > 0$
$\Delta ROI = (\Delta UN / \Delta I) > 0$
Δ: cambio de la variable

Tabla 2. Variación en los indicadores de resultados

Se puede ver claramente que las decisiones en la empresa que modificarán los indicadores operativos deben buscar que tanto la utilidad como el retorno de la inversión aumenten.

1.3. Los cinco pasos para mejorar el sistema

Goldratt propone cinco pasos secuenciales en los que una empresa o institución debe enfocarse y asignar sus esfuerzos para alcanzar los mejores resultados en el mejoramiento del sistema.

Paso 1: identificar la restricción del sistema

Se debe identificar el eslabón más débil de la cadena que está restringiendo el Throughput del sistema. Dependiendo del entorno en que nos encontremos el recurso restricción o cuello de botella será diferente. En un entorno de manufactura es muy probable que la restricción sea una etapa del proceso productivo: una máquina o departamento, donde se esté acumulando inventario en proceso.

Si, por otra parte, se desea identificar la restricción en un proyecto se debe definir la sucesión de actividades más larga del proyecto que determina el tiempo del mismo; a esta sucesión se la denomina la Cadena Crítica. La dependencia entre las actividades está dada por su relación temporal, necesidad de precedencia y el hecho de que el mismo recurso es necesario para su ejecución.

Una vez identificada la restricción del sistema se puede avanzar al paso 3.

Paso 2: Decidir cómo explotar la restricción

Explotar la restricción del sistema se refiere a utilizar al máximo la capacidad del recurso restricción. Se debe aprovechar toda la capacidad disponible de este recurso ya que es éste quien limita el Throughput, y se tiene que hacer sin incurrir en nuevos costos de operación o inversiones.

La administración de la producción o del proyecto debe centrarse en el recurso retracción, con lo cual se puede continuar al paso 3.

Paso 3: Subordinar todo lo demás

Todos los otros componentes del sistema deben actuar de tal manera que garanticen la operación de la restricción al máximo. Los recursos no restricciones son subordinados a la actividad de la restricción. Se prescinde de buscar óptimos locales a lo largo de todos los componentes del sistema y únicamente centrarse en la optimización de la restricción.

Paso 4: Elevar la restricción

Si los pasos 2 y 3 no causaron que la restricción identificada en el paso 1 deje de ser el limitante de Throughput se deberá en este paso elevar la capacidad de la restricción. Se proponen cambios al sistema como: nuevas inversiones de capital, reorganización de los recursos, mejoramiento de los procesos de la restricción o cualquier otra acción que logre hacer que la restricción identificada deje de ser la limitante del sistema. Cuando se ha cumplido el objetivo de elevar la restricción del sistema se debe continuar con el paso 5.

Paso 5: Regresar al paso 1 – Evitar la inercia

La restricción del sistema fue elevada en el paso 3 ó 4 por lo que tenemos que volver al paso 1 e identificar la nueva restricción del sistema y comenzar nuevamente el ciclo de mejora. No se debe permitir que la inercia se apodere de la gestión de la compañía, pues se corre el riesgo de volverse conformista con los logros alcanzados.

Al mejorar los procesos internos se logrará que la restricción salga del sistema y pase a ser la demanda (el mercado) lo que impide generar más Throughput. Aún en estas circunstancias se deberán tomar acciones, obviamente de otra índole, para seguir mejorando el Throughput de la compañía. La línea de meta será empujada hacia adelante cada vez que se alcancen mejoras en el sistema.

Los 5 pasos en los que se debe enfocar la compañía para mejorar el sistema responden a las preguntas: ¿Qué cambiar?, ¿A qué cambiar? y ¿Cómo cambiar? Para decidir qué se debe cambiar, definimos la restricción del sistema. A qué cambiar es definido con las acciones tomadas para explotar la restricción y subordinar el resto de actividades a la operación de la restricción. El cómo será definido en el paso 4 siempre tomando en cuenta los indicadores de operaciones presentados anteriormente.

1.4. Perjuicio de los óptimos locales

En las empresas se ha buscado generalmente optimizar el uso de todos los recursos disponibles para maximizar las utilidades generadas. Como consecuencia lógica de este pensamiento se intenta que la capacidad de la cadena productiva esté balanceada para no tener capacidad ociosa que según los principios de la contabilidad de costos sería una inversión innecesaria. Según la TOC ésta es una práctica que perjudica la generación de Throughput de la compañía y a continuación se presentan los argumentos que respaldan esta afirmación.

Un sistema por naturaleza no tiene capacidad balanceada ya que cada elemento realiza tareas distintas con diferentes niveles de complejidad y por esta razón resulta casi imposible tener eficiencias locales máximas en todos los procesos, salvo con la creación de inventario en proceso innecesario. Con el afán de reducir costos se busca mecanismos para balancear la capacidad y no tener tiempos ociosos.

Una cadena balanceada afronta dos hechos que afectan su generación de Throughput y la rentabilidad de la operación: 1) las fluctuaciones estadísticas y 2) la

dependencia entre los procesos. Un ejemplo simplificado con tres recursos que tienen una capacidad promedio de 5 unidades por hora demuestra las consecuencias negativas.

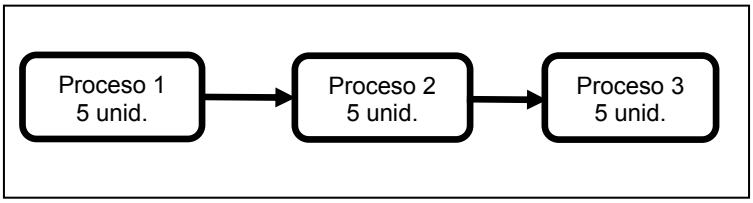

Figura 2. Cadena balanceada

Cada proceso de la cadena tiene una desviación estándar máxima de 1.4 unidades y cada proceso depende del material entregado por el proceso anterior. Al final de un día de trabajo de 8 horas en teoría la cadena debería producir 40 unidades. En la tabla 3 se resumen los resultados de un posible día de trabajo.

Horas	Proceso 1	Proceso 2			Proceso 3			Resultado
	Real	IP	Real	Potencial	IP	Real	Potencial	
1	4	0	4	5	0	4	5	4
2	5	0	5	5	0	5	6	5
3	7	3	4	4	0	4	6	4
4	5	2	6	6	2	4	4	4
5	4	0	6	6	4	4	4	4
6	3	0	3	4	0	7	7	7
7	5	0	5	5	1	4	4	4
8	7	2	5	5	2	4	4	4
Promedio	**5**		**4.75**	**5**		**4.5**	**5**	**4.5**
Desv. Est.	**1.4**		**1.0**	**0.8**		**1.1**	**1.2**	
Total	**40**		**38**	**40**		**36**	**40**	**36**

IP = Inventario en proceso

Tabla 3. Producción balanceada[9]

Como se puede ver las fluctuaciones en los tiempos de los procesos en algunas ocasiones no se promedian debido a que no se dispone de material proveniente del proceso anterior, la capacidad potencial es superior a la cantidad producida. Los atrasos se acumulan de proceso en proceso dando como resultado que la producción real no sea la esperada (36 unidades producidas de 40 esperadas) y se acumule inventarios en

[9] Thomas Corbett, *Throughput Accounting*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1998, pág. 171.

proceso. Mientras más etapas existan en la cadena el impacto en los resultados finales será mayor, y también si las fluctuaciones son más pronunciadas.

Los esfuerzos para balancear la cadena de un sistema son infructuosos ya que perjudican al Throughput generado y para cumplir con la producción esperada se tiene que recurrir a proporcionar más capacidad a los recursos retazados e incurrir en costos de horas extras, o en el peor de los casos ocasionar atrasos en las entregas a los clientes. La búsqueda de óptimos locales envés de reducir costos pone en peligro el Throughput y aleja al sistema de su meta.

1.5. Administración de la cadena de Throughput

Para administrar la cadena de Throughput debemos seguir los 5 pasos especificados anteriormente. Se describirán con detalle la forma de aplicarlos en un ambiente productivo industrial, que fue el principio de la TOC, y su adaptación a la administración general de proyectos. Posteriormente en el capítulo 3 de esta tesis se presentará la aplicación de los 5 pasos para proyectos específicos de software.

Administración de la producción industrial – La metodología DBR

La metodología DBR[10] (Tambor-Amortiguador-Cuerda) es la aplicación de la TOC para sincronizar la producción de una planta industrial. La teoría de restricciones centra su enfoque en el recurso con capacidad limitada, por lo que es este componente el primer elemento que ayuda a planificar la producción. El recurso restricción del sistema puede procesar la materia prima a una tasa dada (se asume que el proceso se encuentra bajo control estadístico), a esta tasa de producción se hace la analogía con el ritmo de un tambor. La tasa de producción del recurso restricción marca el paso de todos los

[10] DBR son las siglas en inglés de “Drum-Buffer-Rope” que significa Tambor-Amortiguador-Cuerda. En este estudio se respetará el término técnico DBR.

componentes del sistema productivo: la restricción es el *tambor*. La capacidad del recurso restricción permitirá generar un cronograma detallado de producción.

El paso tres nos sugiere que los demás componentes del sistema, que son la mayoría, deben trabajar subordinados a la capacidad del recurso restricción. Goldratt nos dice que los recursos no restricción deben trabajar con una política de "correcaminos": trabajar a su máxima velocidad cuando se necesite y detenerse cuando no son necesarios. Esta política de subordinación es conocida como la *cuerda*. Los recursos no restricción son "amarrados" al avance de la restricción del sistema.

La velocidad con la que se libera materia prima para ser procesada debe estar sincronizada con la capacidad de consumirla del recurso restricción. La liberación de materia prima está dada por el tiempo que tarda en llegar al recurso restricción a través de los procesos previos. Este criterio debe mantenerse aunque algunos componentes del sistema tengan que permanecer algún tiempo sin trabajar. Esto se lo hace principalmente para no incrementar innecesariamente el inventario de productos en proceso y aumentar la cantidad de inversión atrapada en el sistema.

Los lotes de materia prima liberados a ser procesados por el recurso restricción se los organiza según las fecha de entrega programada de los productos terminados: los lotes con fechas más próximas serán procesados primero.

El último componente de la metodología DBR es la administración del *amortiguador*. El uso de un amortiguador es vital para proteger al recurso retracción de posibles perturbaciones en los procesos y la variabilidad natural inherente en los recursos no restricción. Debemos recordar que una hora perdida en el recurso restricción es una hora perdida de todo el sistema. Se debe garantizar que la restricción no pare su producción por posibles problemas en otros componentes del sistema.

La unidad de medida del amortiguador frente al recurso restricción es tiempo y se establece su duración basándose en la variabilidad de los procesos que pueden afectar el suministro de material al recurso restricción. Mientras más grande es la variabilidad de los procesos más grande será el amortiguador y viceversa.

Administración de proyectos – La metodología de Cadena Crítica

Los proyectos son divididos en etapas y cada etapa está conformada por actividades a realizarse en un orden lógico. La planificación de estas actividades y su permanente control son tareas fundamentales para el éxito de los proyectos.

Un proyecto se considera exitoso si cumple con tres requisitos: i) cuando las características y funcionalidades del producto final satisfacen todos los requerimientos inicialmente solicitados; ii) el tiempo de ejecución estuvo acorde al cronograma establecido; y, iii) cuando el presupuesto aprobado fue respetado, siendo este factor de gran importancia.

En la realidad el porcentaje de proyectos que se pueden considerar exitosos es muy bajo ya que no cumplen por lo menos uno de los criterios mencionados. A pesar de la tendencia a inflar los tiempos de las actividades del proyecto, estos no se logran terminar dentro del cronograma.

Los problemas más comunes que causan atrasos en los proyectos son el "síndrome del estudiante" y la variabilidad e interdependencia de las actividades. El síndrome del estudiante describe la tendencia de poner énfasis en la culminación de una actividad o tarea sólo cuando se acerca la fecha de entrega. La figura 3 ilustra el problema.

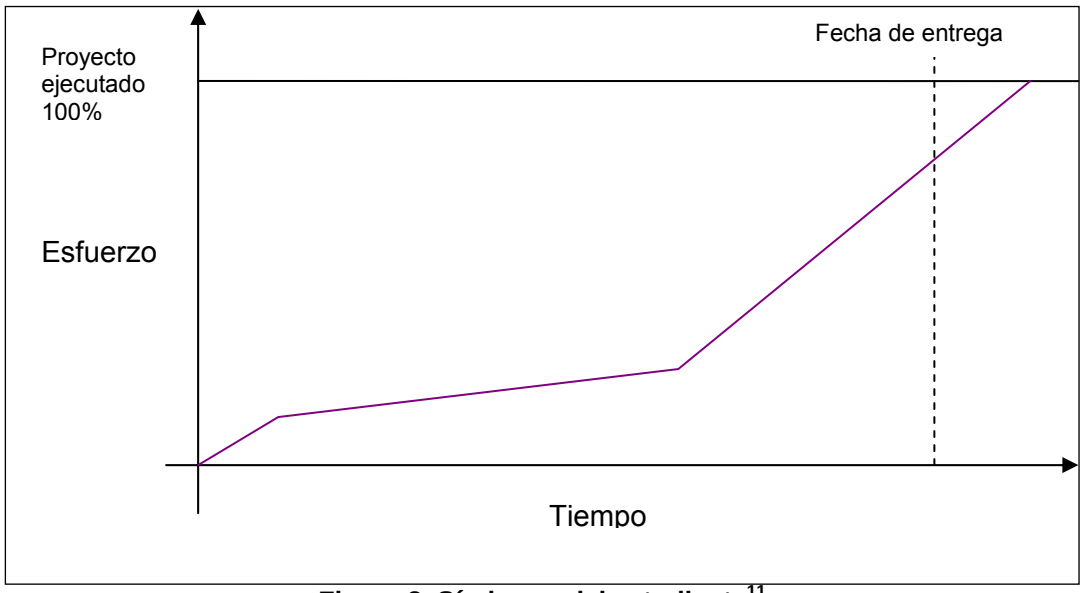

Figura 3. Síndrome del estudiante[11]

El segundo problema se presenta por la variabilidad natural en el tiempo de culminación de las actividades y la interdependencia entre ellas. Generalmente si una tarea es culminada antes de los pronosticado la tarea siguiente no es empezada con anticipación y se desperdicia el tiempo ganado. Sin embargo, cuando una tarea toma más tiempo del establecido el atraso se propaga a las tareas siguientes. Esta práctica ocasiona que no se acumulen los tiempos ganados cuando las tareas se terminan antes y sólo se acumulen los atrasos, las variabilidades negativas y positivas no son compensadas y el tiempo del proyecto total es afectado.

Para administrar eficientemente los proyectos, la TOC indica que no es importante proteger el tiempo de culminación de las actividades individuales y colocar colchones de tiempo en cada una, lo importante es asegurar la culminación total del proyecto en la fecha prometida, por esto, es vital proteger al proyecto en su conjunto con un único colchón de tiempo que absorba las desviaciones y posible perturbaciones.

El primer paso para proteger un proyecto es precisar la cadena crítica. Definimos la cadena crítica como la secuencia más larga de actividades dependientes ya sea por uso

[11] Domenico Lepore y Oded Cohen, *Deming and Goldratt The Theory of Constraints and The System of Profound Knowledge*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1999, pág. 74.

de recursos o precedencia lógica y el amortiguador establecido. Al hacer esto se ha encontrado la restricción, ya que la cadena crítica determina el tiempo total necesario para el proyecto y la habilidad para generar Throughput.

Para explotar la restricción es necesario garantizar que las tareas pertenecientes a la cadena crítica no retrasen la fecha de culminación del proyecto. Goldratt señala que no es importante proteger la fecha de culminación de cada actividad, lo que se debe garantizar es la fecha de culminación del proyecto en su conjunto. Para lograr esto no se debe poner holguras de tiempo a las actividades, sino un amortiguador de tiempo que proteja al proyecto de las variabilidades naturales de las actividades. De esta manera se evita el "síndrome del estudiante" y que se desperdicie tiempo valioso para el proyecto.

El resto de actividades y secuencias que no pertenecen a la cadena crítica, conocidas como cadenas de alimentación, deben ser programadas subordinadas a las necesidades de su culminación según la cadena crítica. Nuevamente, para proteger a la cadena crítica no se ponen holguras a las actividades sino a la cadena de alimentación en su conjunto. El siguiente gráfico (figura 4) explica lo expuesto.

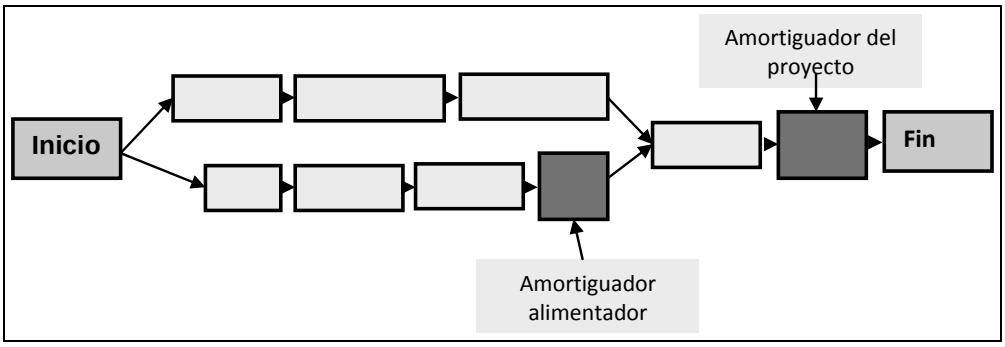

Figura 4. La Cadena Crítica[12]

[12] David Anderson, *Agile management for software engineering*, Estados Unidos de América, Editorial Prentice Hall PTR, 2004, pag. 70.

CAPITULO 2.
DESARROLLO DE SOFTWARE Y LAS METODOLOGIAS ÁGILES

2.1. El desarrollo de software y su evolución

Hasta hace medio siglo nadie se hubiera podido imaginar que el software se convertiría en una tecnología indispensable para las actividades humanas y que tendría un impacto tan dramático en los negocios, la ciencia, las comunicaciones, la ingeniería. El desarrollo de software es el motor detrás de la revolución de las computadoras personales y se ha constituido en un factor vital en la economía. “En la actualidad, el software es la tecnología individual más importante en el ámbito mundial”.[13]

Conforme la industria del software ha ido evolucionando los desarrolladores se han preocupado por mejorar la base tecnológica existente para hacer más fácil el desarrollo, puesta en marcha y mantenimiento de los programas de software. Han evolucionado los frameworks de desarrollo y los programadores cuenta cada vez con herramientas más sofisticadas para trabajar. Pese a esto se sigue evidenciado retrasos en los proyectos y falta de cumplimiento de los presupuestos. Esta tesis se centra en el análisis de cómo mejorar la administración de los proyectos de software para entregar un producto de calidad de una manera rentable.

Los proyectos de desarrollo e implementación de software tienen una estructura macro similar a proyectos en otras áreas, por ejemplo un proyecto de la industria de la construcción o el desarrollo de un nuevo producto en una industria siguen los mismos parámetros para su administración que un proyecto relacionado con el desarrollo de software. Los proyectos son divididos en etapas y cada etapa está conformada por actividades a realizarse en un orden lógico. La planificación de estas actividades y su permanente control son tareas fundamentales para el éxito de los proyectos y no es una

[13] Roger Pressman, *Ingeniería del Software*, 6ta Edición, México, McGraw-Hill, 2006. pág. 1

excepción en el desarrollo de software. Pese a estas similitudes se debe tomar en cuenta que el software es un elemento lógico y no físico, lo que conlleva una serie de diferencias que serán analizadas.

Un proyecto de desarrollo de software se considera exitoso si cumple con tres requisitos: i) cuando las características y funcionalidades del producto final satisfacen todos los requerimientos inicialmente solicitados; ii) el tiempo de desarrollo estuvo acorde al cronograma establecido; y, iii) cuando el presupuesto aprobado fue respetado, siendo este factor de gran importancia. Las estadísticas elaboradas por The Standish Group revelan que sólo un 29% de los proyectos de tecnologías de la información (IT) implementados mundialmente cumplen con las tres características para ser considerados exitosos.[14]

Las técnicas tradicionales para la administración de proyectos como el Método de la Ruta Crítica (CPM) y la Técnica de Revisión y Evaluación de Programas (PERT) han sido utilizadas por décadas para realizar las actividades antes mencionadas, sin embargo en los proyectos de desarrollo de software el problema más común es el desfase del cronograma establecido. Esto evidencia que las metodologías utilizadas no son adecuadas o suficientes.

Los retrasos en los cronogramas de los proyectos de desarrollo de software se presentan generalmente por las siguientes causas:

- falta de coordinación entre los involucrados en el proyecto
- la ineficiencia en el uso de los recursos disponibles que se ocasiona por no implementar metodologías de desarrollo adecuadas y por no tener un mejor control
- por la mala elaboración, subestimación o cambios de los requerimientos inicialmente establecidos

[14] The Standish Group International, Inc, "*2004 Third Quarter Research Report*", http://standishgroup.com/quarterly_reports.

- por la utilización de tecnologías no probadas y por emplear desarrolladores sin la experiencia o formación adecuada.
- mala gestión administrativa y financiera

Estos problemas causan demoras y malestar en los usuarios finales del producto y pérdidas económicas para las empresas desarrolladoras.

El desarrollo de software es un proceso de continuo aprendizaje y mejora. Para administrar y controlar este proceso de desarrollo se han utilizado modelos para crear un orden, un marco de trabajo. El *modelo de cascada*, o *ciclo de vida clásico*, ha sido utilizado por décadas. Este modelo fue descrito por primera vez de una manera formal por Winston Royce en 1970.[15] Este modelo propone un trabajo sistemático y secuencial con las siguientes etapas: levantamiento de requerimientos del cliente; análisis, modelado y planeación; desarrollo y pruebas de unidad; implementación y pruebas del sistema; y concluye con el soporte proporcionado al software terminado. Las falencias de este modelo que se enunciarán a continuación han hecho que sea muy cuestionado por los desarrolladores y la industria del software en general.

El principal problema del modelo clásico de cascada es que asume que los requerimientos iniciales del proyecto están completos, detallados correctamente y no cambian. Esto ocasiona que resulte muy difícil y costoso incorporar cambios o nuevos requerimientos en etapas avanzadas en el desarrollo. En proyectos reales es improbable que el cliente/usuario pueda definir la totalidad de requerimientos de manera explícita, por lo que los proyectos siempre tendrán un grado de incertidumbre.

[15] Winston Royce (1929–1995) fue un destacado ingeniero en sistemas informáticos. Fue director del Lockheed Software Technology Center en Austin, Texas, Estados Unidos de América. Royce no utilizó el término *modelo de cascada* en su artículo de 1970 ni lo propuso cómo un modelo óptimo para el desarrollo de software.

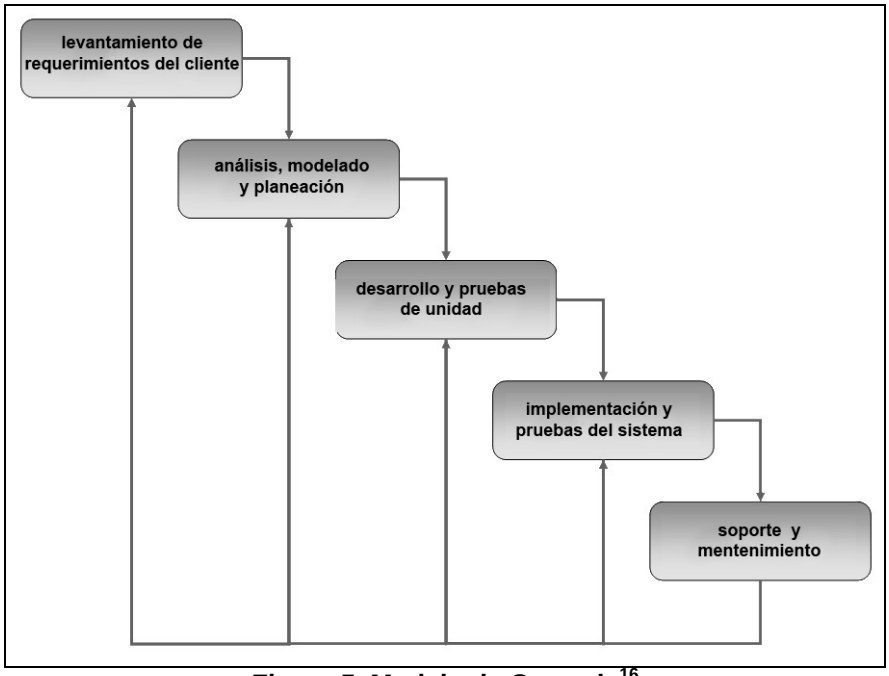

Figura 5. Modelo de Cascada[16]

En un intento por superar las falencias en el desarrollo de software otros modelos fueron propuestos con la premisa de tener un desarrollo de software por iteraciones. Cada iteración implica la entrega de un software funcional que será mejorado en la siguiente entrega. Entre los modelos que siguen este enfoque están: el modelo incremental, que adapta el modelo de cascada y divide el proyecto en varias fases; el modelo de espiral que propone un desarrollo evolutivo con las primeras versiones en estado de prototipo y las siguientes iteraciones liberarán versiones más estables y complejas.

Un nuevo enfoque para el desarrollo de software fue propuesto con las metodologías de Desarrollo Ágil, que han tenido un gran auge en la última década. Estas metodologías también proponen un desarrollo por iteraciones propiciando la liberación de versiones funcionales en tiempos más cortos que el modelo de cascada. Se profundizará el análisis de estas metodologías en el siguiente apartado.

[16] Michele Sliger y Stacia Broderick, *The Software Project Manager's Bridge to Agility*, Estados Unidos de América, Addison-Wesley Professional, 2008, pág. 29.

2.2. Las metodologías Ágiles de Desarrollo

Desde mediados de la década de los noventa han evolucionado las denominadas metodologías de Desarrollo Ágil para enfrentar los problemas en el desarrollo de software. Estas metodologías Ágiles promueven una nueva filosofía y un conjunto de directrices que se enfocan principalmente en un proceso de varias iteraciones durante el desarrollo de un proyecto, permitiendo que los tiempos de desarrollo sean más cortos ya que los cambios y mejoras se pueden hacer en etapas tempranas, logrando que el producto final esté más acorde a las necesidades de los usuarios y por ende sea de mejor calidad.

En 2001 un grupo de notables desarrolladores de software crearon el manifiesto para el desarrollo ágil de software:[17]

> Nosotros estamos descubriendo mejores formas para el desarrollo de software al irlo haciendo y al ayudar a otros a hacerlo. A través de nuestro trabajo hemos llegado a valorar:
>
> A **los individuos y sus interacciones** sobre los procesos y las herramientas
>
> Al **software funcional** sobre la extensa documentación
>
> A **la colaboración del cliente** sobre la negociación del contrato
>
> A **la respuesta al cambio** sobre el seguimiento del plan
>
> Esto es, aunque tienen valor los elementos de la derecha, valoramos más los elementos de la izquierda[18] (resaltados con negrilla).

Las metodologías Ágiles de desarrollo no son la antítesis de los principios tradicionales de la ingeniería de software, lo que se busca es adaptar esos valiosos conceptos, métodos y herramientas para obtener mejores resultados con condiciones

---

[17] La definición hasta ahora aceptada de las metodologías Ágiles de desarrollo de software fue elaborada por 17 profesionales en febrero de 2001 en una convención de metodologías de desarrollo en Utah, Estados Unidos de América. En ese tiempo eran conocidas como las metodologías "livianas" de desarrollo. Entre las metodologías Agiles se encuentra XP (Extrema programmig), FDD (Feature-Driven Development), Scrum, entre otras.

[18] Traducido de http://agilemanifesto.org/ Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas

cambiantes del mercado y ajustarse a las necesidades del cliente y el entorno del negocio reinante. Es importante recalcar que las metodologías Ágiles ponen énfasis en los elementos de la izquierda del manifiesto pero sin desacreditar los elementos de la derecha, y estos últimos son utilizados al nivel que añadan valor al proyecto.

Las metodologías Ágiles a más de incorporar flexibilidad en el proceso de desarrollo delegan gran parte de la responsabilidad del proyecto al equipo desarrollador, por lo que estimulan la creación de equipos capacitados, con actitudes proactivas y una comunicación constante entre sus miembros. Se resalta la entrega rápida de software funcional, el cual continuará evolucionado, y un punto crítico es que se involucra al cliente/usuario como parte del equipo de desarrollo.

El manifiesto Ágil fue escrito junto con doce principios para guiar al equipo de desarrollo durante los proyectos, resulta de gran utilidad tener presente estos principios, ya que sirven como norte en momentos de problemas o dudas. Estos doce principios son los enumerados a continuación:[19]

1. Nuestra máxima prioridad es satisfacer al cliente a través de entregas rápidas y continuas de software valioso.
2. Le damos la bienvenida a los requerimientos cambiantes, incluso aunque aparezcan tarde en el desarrollo. Los procesos Ágiles aprovechan al cambio para crear una ventaja competitiva en el cliente.
3. Entregamos software funcional frecuentemente, entre un par de semanas a un par de meses, prefiriendo los tiempos más cortos.
4. Las personas del negocio (usuario/cliente) y los desarrolladores deben trabajar juntos diariamente a lo largo de todo el proyecto.

[19] Traducido de http://agilemanifesto.org/ Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas.

5. Construimos proyectos sobre individuos motivados. Les damos el ambiente y el apoyo que necesitan, y confiamos en ellos para hacer el trabajo.
6. El método más eficiente y efectivo de recolectar información para y dentro de un equipo de desarrollo es la conversación cara-a-cara.
7. La principal métrica de avance es el software funcional.
8. Los procesos Ágiles fomentan el desarrollo sustentable. Los auspiciantes, los desarrolladores y los usuarios deberían ser capaces de mantener de un ritmo constante de forma indefinida.
9. Mejoramos la agilidad prestando continua atención a la excelencia técnica y al buen diseño.
10. La simplicidad - el arte de maximizar la cantidad de trabajo que no se hace - es esencial.
11. Las mejores arquitecturas, requerimientos y diseños emergen de los equipos auto-gestionados.
12. A intervalos regulares, el equipo reflexiona sobre cómo ser más efectivo, y luego ajusta su comportamiento de forma acorde.

Una vez conocidos los principios que gobiernan las metodologías Ágiles se puede profundizar en el proceso de desarrollo de software. Conforme lo señala Roger S. Pressman[20], este proceso incorpora las siguientes premisas en la administración de los proyectos:

- Es difícil establecer desde un inicio los requerimientos totales del software y el detalle de las funcionalidades.
- El diseño y desarrollo deben ser intercalados, por eso se propone un proceso de iteraciones incrementales en cortos períodos de tiempo.

[20] Roger S. Pressman, *Ingeniería del Software, un enfoque práctico*, México, McGraw-Hill, 2006, pág. 81.

- Existe un grado de incertidumbre en el análisis, diseño y desarrollo por lo que se requiere adaptabilidad y retroalimentación constante del cliente/usuario.

Un proceso ágil de desarrollo requiere una adaptación continua, para lograr esto, como se mencionó anteriormente, es necesario de la retroalimentación constante del usuario/cliente basándose en prototipos entregados de manera incremental. Todas las metodologías de desarrollo Ágil incorporan una estrategia de iteraciones con software funcional como entregable de cada fase. El cliente/usuario de esta manera puede evaluar el producto de manera regular y proporcionar retroalimentación valiosa al equipo e influir en el proceso de desarrollo.

En el siguiente apartado se profundizará en este tema basándose en la metodología de desarrollo Scrum.

2.3. La metodología Ágil de desarrollo Scrum

"Scrum es un proceso iterativo e incremental para desarrollar cualquier producto o administrar cualquier trabajo. Él produce un conjunto de funcionalidades potencialmente entregables al final de cada iteración."[21] Scrum fue creado por Jeff Sutherland y Ken Schwaber[22] y sus principios son consistentes con los del manifiesto Ágil.

Scrum se enfoca en formar un dedicado equipo de desarrollo, interdisciplinario y auto administrado. El equipo es responsable de desarrollar un producto de software de manera incremental incluyendo al usuario/cliente en el proceso.

Una gran ventaja de Scrum es que es un marco de trabajo bastante simple; el proceso, las herramientas y reglas son pocas y fáciles de aprender. Es extremadamente útil para facilitar cumplir con un trabajo complejo en el cual es imposible predecir todo lo que va a pasar y definir a detalle los requerimientos y funcionalidades del trabajo

[21] Ken Schwaber, "What is Scrum?", www.controlchaos.com.
[22] Sutherland y Schwaber se basaron en las ideas presentadas en el artículo "The New New Product Development Game" (Harvard Business Review 86116:137-146, 1986) escrito por Ikujiro Nonaka and Hirotaka Takeuchi

terminado. Scrum ayuda a mantener al equipo informado sobre los detalles y avances del proyecto, y esto permite que sus miembros tomen decisiones y ajustes inmediatos para guiar al proyecto hacia la meta deseada.

Existen tres roles bien definidos dentro de Scrum: el dueño del producto, quien es el responsable por el éxito del proyecto, definir las funcionalidades a desarrollarse y representar los intereses de todos los involucrados en el proyecto; el equipo de desarrollo, que está formado por profesionales interdisciplinarios encargados del desarrollo del producto, se recomienda que el equipo esté formado por un máximo de 6 a 8 integrantes;[23] y el Scrum Master, que es la persona encargada de la correcta ejecución del proceso de Scrum, sirve de guía y líder del equipo de desarrollo y facilita la comunicación entre el dueño del producto y el equipo.

La metodología Scrum ayuda a guiar las actividades inherentes del proceso de desarrollo de software, estas actividades son: levantamiento de requerimientos, análisis, diseño de la solución tecnológica, desarrollo, control de calidad y entrega. En cada interacción, llamada *sprint*, se llevan a cabo generalmente las 6 actividades mencionadas. La metodología sugiere que cada *sprint* dure 30 días calendario, en consecuencia cada 30 días se entrega un conjunto de nuevas funcionalidades, del producto en desarrollo al usuario/cliente.

El tiempo que se asigna a cada *sprint* puede variar dependiendo de tiempo total que demorará cada proyecto. Una buena sugerencia sería dividir el proyecto en por lo menos 4 *sprints*, así, si se está administrado un proyecto de dos meses de duración, se podrán tener *sprint* de 15 días. Se decidirá que *sprints* proporcionarán funcionalidades a ser instaladas para el uso del los clientes.

Al inicio de cada *sprint* se realiza una reunión de planificación de la iteración de una duración máxima de 8 horas. En las primeras 4 horas se decide las nuevas

[23] Si el proyecto a desarrollarse requiere de un equipo más grande de desarrolladores se sugiere que se divida el proyecto total en sub proyectos y cada uno tenga su propio equipo definido.

funcionalidades a ser desarrolladas y entregadas dentro de 30 días. Se da prioridad a las características o funcionalidades que mayor valor entregue al usuario/cliente. Para definir la cantidad de funcionalidades a ser desarrolladas en el *sprint* se debe calcular el número de horas laborables disponibles en la ventana de tiempo propuesta y el número de desarrolladores involucrados en el proyecto. Cada funcionalidad debe ser dividida en actividades de una duración máxima de 16 horas para poder estimar con mejor precisión la cantidad de trabajo que podrá ser culminado. La lista de funcionalidades del *sprint* queda registrada y se convierte en un compromiso a cumplir por el equipo.[24]

La segunda parte de la reunión sirve para planificar las actividades a ser desarrolladas por el equipo dentro del marco de tiempo establecido y organizar el trabajo. Luego de esta reunión de inicio del *sprint* empiezan a contar los 30 días disponibles, o el tiempo que se haya definido.

Todos los días el equipo de desarrollo completo mantiene una reunión de 15 minutos liderada por el Scrum Master a la misma hora y en el mismo lugar, llamada el Scrum diario. Esta reunión debe mantener siempre su enfoque y cada miembro del equipo debe responder únicamente estas 3 preguntas:

- ¿Qué has hecho desde la última reunión?
- ¿Qué dificultades u obstáculos encontraste?
- ¿Qué planeas hacer hasta la próxima reunión?

El Scrum diario sirve para monitorear el avance del proyecto, identificar las posibles dificultades encontradas por el equipo, definir el trabajo a ser realizado durante el día y compartir el conocimiento adquirido entre los miembros del equipo. Si durante la reunión se ve la necesidad de profundizar sobre algún tema del proyecto, las personas

[24] Tanto el tiempo designado para la reunión de planificación cómo la duración de las tareas deben estar acordes a la duración del sprint y mantener una proporcionalidad lógica con los tiempos sugeridos para los sprint de 30 días.

involucradas deben acordar una reunión después de concluido el Scrum diario para su discusión.

Scrum utiliza una herramienta llamada el *burndown chart* para monitorear la cantidad de trabajo pendiente por ser realizado en el *sprint*. Esta herramienta ayuda a visualizar las horas de trabajo necesarias todavía para finalizar las funcionalidades seleccionadas en comparación con los días restantes del plazo definido. En el eje Y se coloca el número de días que se estima tomará en desarrollarse todas las funcionalidades faltantes y el eje X corresponde al día en que se encuentra el *sprint.* Este gráfico es preparado diariamente por el Scrum Master y ayuda a evidenciar si el ritmo de trabajo está acorde con lo planeado. Un ejemplo se presenta en la figura 6.

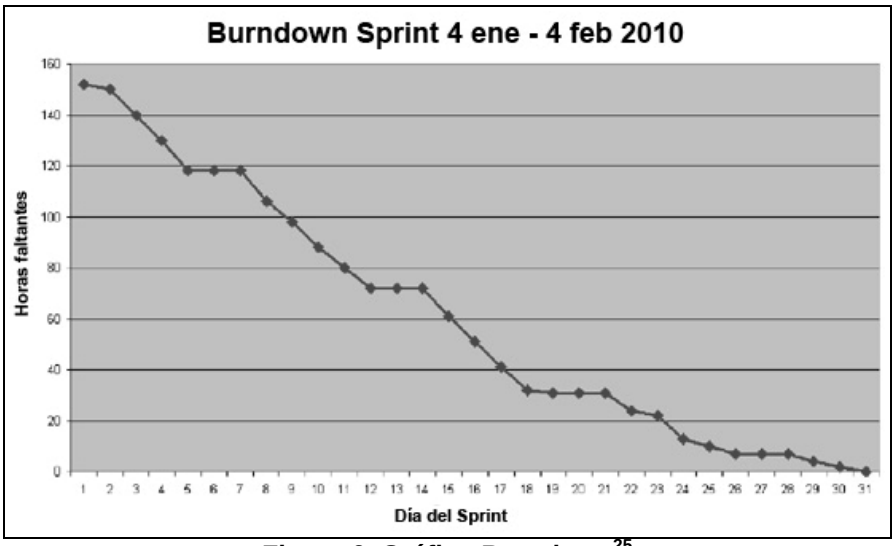

Figura 6. Gráfico Burndown[25]

El *burndown chart* es actualizado con los informes diarios proporcionados por los miembros del equipo en el *sprint* diario. Cada miembro reporta las funcionalidades que se encuentran en estado de "listo". Es muy importante que la definición del "listo" sea dejada muy en claro por el Scrum Master y entendida por todos los miembros del equipo.

[25] Ken Schwaber, Agile Project Management with Scrum, Estados Unidos de América, Microsoft Press, 2004, pág. 12.

Generalmente no se reporta una funcionalidad en estado de “listo” hasta ésta ha sido probada y ha pasado por el control de calidad establecido.

A la finalización de cada *sprint* se lleva a cabo la reunión de revisión. Esta reunión debe tener una duración máxima de 4 horas y es cuando el equipo de desarrollo presenta las nuevas funcionalidades desarrolladas al dueño del producto y a las personas relacionadas con el proyecto que deseen asistir. En esta reunión se recibe retroalimentación que será útil para definir los trabajos a desarrollarse en el siguiente *sprint* y poder incorporar mejoras al proceso sugeridas que ayuden a ser más eficiente la siguiente iteración. En la figura 7 se resume el proceso de Scrum.

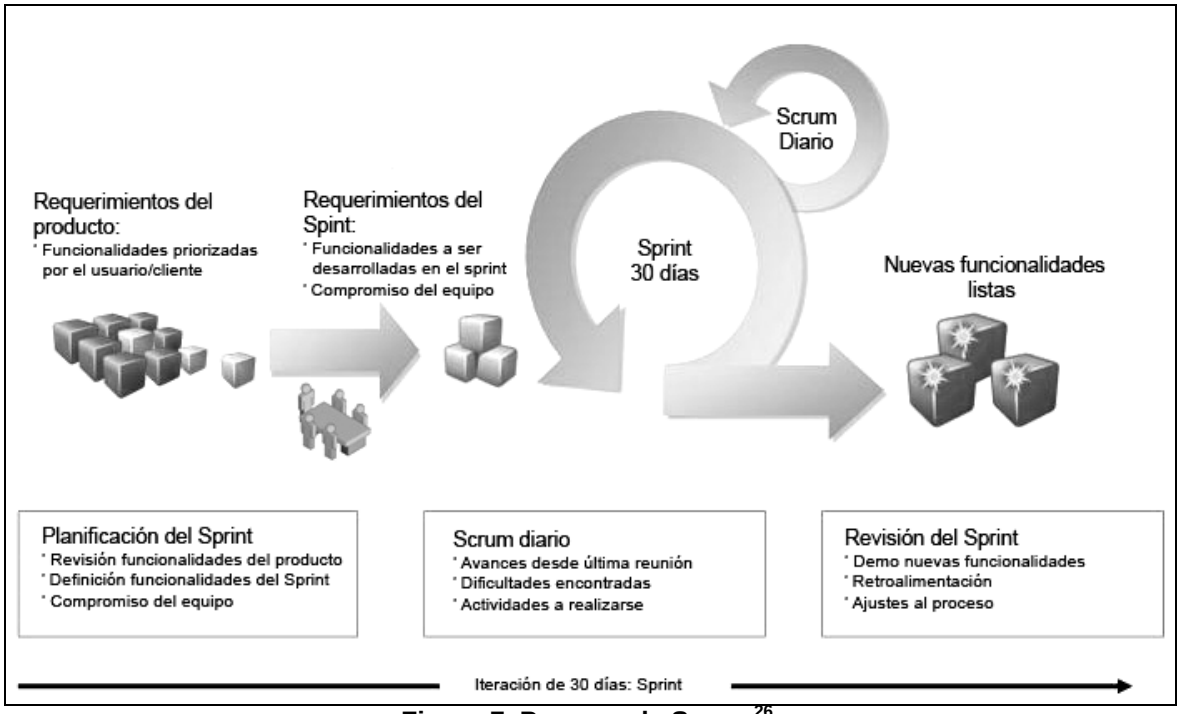

Figura 7. Proceso de Scrum[26]

[26] Ken Schwaber, Agile Project Management with Scrum, Estados Unidos de América, Microsoft Press, 2004, pág. 9.

CAPITULO 3.
TOC EN EL DESARROLLO DE SOFTWARE

3.1. Particularidades del desarrollo de software y la meta del sistema

En el primer capítulo se estableció que la meta de una empresa con fines de lucro es ganar dinero en el presente y en el futuro. Por esta razón, para una empresa de desarrollo de software o un departamento de desarrollo no es suficiente hacer buen código y software de calidad, es esencial que cada proyecto tenga un adecuado retorno sobre la inversión realizada. El negocio de desarrollo de software, como cualquier otro, debe enfocarse en generar mayores utilidades y elevar su rendimiento, y se recuerda que según la TOC se debe buscar:

- Incrementar el Throughput (T)
- Reducir la Inversión e Inventarios (I)
- Reducir los Costos de Operación (OE)

Los términos utilizados por TOC se explicarán a continuación en relación con la industria de desarrollo de software:

- Throughput: Es la tasa a la que se genera efectivo a través de la entrega de software funcional. No se considera Throughput código o software terminado. Sólo cuando el cliente ha realizado el pago por los entregables se ha generado Throughput.
- Inversión: La cantidad de dinero invertido en la empresa de desarrollo de software necesario para su funcionamiento, más la cantidad invertida para obtener los requerimientos demandados por los clientes/usuarios que sirven de insumos para la producción de software.
- Inventario: Todos los requerimientos demandados por los clientes/usuarios que se encuentran en espera de ser convertidos en funcionalidades del software, o que se encuentran en desarrollo, o que aunque estén terminados no han sido convertidos en efectivo.

- Costos de Operación: Es la cantidad de dinero gastada para convertir los requerimientos de los clientes/usuarios en software funcional.

Para lograr incrementar el Throughput se debe lograr entregar software que agregue valor a los clientes/usuarios con mayor frecuencia. Se asume que la restricción para lograrlo no se encuentra en el mercado o en la habilidad de la empresa para concretar ventas,[27] sino es una restricción interna que puede ser explotada. El recurso restricción en el proceso de desarrollo de software frecuentemente es uno o un grupo de desarrolladores senior especialistas, como por ejemplo diseñadores de interfases, arquitectos o modeladores de datos.

Aplicando la metodología de desarrollo Ágil Scrum se puede mejorar el Throughput de la empresa, ya que como se explicó en el capítulo 2, Scrum se enfoca en tener entregas constantes de software funcional cada 30 días. Estas entregas frecuentes y constantes permitirán a la empresa desarrolladora facturar los entregables y convertir con mayor rapidez en Throughput los requerimientos demandados por los clientes/usuarios.

Reducir la Inversión significa que el costo incurrido en el levantamiento de requerimientos del cliente debe ser menor aplicando mejores formas de identificar las funcionalidades de mayor valor demandadas, y comunicar éstas a los desarrolladores. En este caso, Scrum es también de gran ayuda ya que involucra al cliente en el proceso de desarrollo y en la creación y priorización de la lista de requerimientos del producto. Se recuerda que Scrum aboga por que desde el inicio del proyecto el cliente tenga un rol activo y entienda que sin su involucramiento el proyecto no será exitoso. Gracias a la constante colaboración del los clientes/usuarios en este proceso el tiempo y dinero invertido será menor y se mitiga el riesgo de la incertidumbre en el desarrollo. Un punto

[27] Goldaratt ha elaborado herramientas para explotar las restricciones externas a la empresa en las áreas de ventas y mercadeo, éstas se detallan en su libro *No fue la suerte*, pero ese análisis sale del alcance de esta tesis.

adicional de suma importancia es que la inversión en el levantamiento de requerimientos se la hace parcialmente para cada iteración y es dividida a lo largo de todo el proyecto y no en su totalidad al inicio del mismo. La lista de requerimientos del producto contiene todas las funcionalidades a ser desarrolladas, pero sólo cuando son incluidas en el *sprint* actual son analizadas a profundidad. Esto ayuda a tener una idea global del proyecto pero a enfocarnos principalmente en las funcionalidades a ser desarrolladas al corto plazo.

Las mismas ventajas de Scrum que ayudan a reducir la inversión también hacen que los inventarios permanezcan bajos. Se recuerda que según la TOC que en el caso de desarrollo de software el inventario es considerado todos los requerimientos del cliente que han sido analizados y están a la espera de ser trasformados en software funcional y que el pago correspondiente. Los *sprints* de 30 días hacen que se tenga inventario únicamente para ese período y potencialmente que ese inventario se transforme en Throughput mensualmente.

Los Costos de Operación también son muy bien controlados al aplicar Scrum. El principio diez de las metodologías Ágiles dice: La simplicidad - el arte de maximizar la cantidad de trabajo que no se hace - es esencial. Las funcionalidades que son valoradas por los clientes/usuarios son las únicas que deben ser desarrolladas. Tiempo y dinero gastado en el desarrollo de funcionalidades no pedidas o que no agregan valor se convierte en desperdicio. En cada *sprint* se priorizan las funcionalidades a ser desarrolladas y liberadas a los clientes/usuarios. El principal componente en los Costos de Operación en el desarrollo de software son los sueldos y beneficios pagados a los desarrolladores, por este motivo se enfatiza que las horas trabajadas sean utilizadas únicamente en funcionalidades que ayuden a incrementar el Throughput.

3.2. Los cinco pasos para mejorar el desarrollo de software

Se resume a continuación los cinco pasos propuestos por la TOC para optimizar un sistema:

1. Identificar la restricción del sistema
2. Decidir cómo explotar[28] la restricción
3. Subordinar todo lo demás
4. Elevar la restricción
5. Regresar al paso 1 – Evitar la inercia

Se recuerda que la TOC está fundamentada en el principio de que una cadena o sistema es tan fuerte como su eslabón más débil y que existe sólo un eslabón más débil, conocido como el recurso restricción o cuello de botella, en un momento determinado. Por esto resulta obvio que en nuestro proceso de mejora se debe identificar como primer paso el recurso restricción que está limitando el Throughput de la compañía.

Podría parecer intuitivo que la restricción del sistema de desarrollo de software son las 8 horas al día que tienen los desarrolladores, o siendo más conservadores se pueden tomar 6 ó 7 horas al día de trabajo realmente productivo. Pero el tiempo de que disponen los desarrolladores no es en realidad la restricción, se debe definir el recurso que esté haciendo de cuello de botella en el sistema. Generalmente en la cadena se acumula inventario adelante del cuello de botella y después de él se carece de insumos para el resto de procesos.

Se recuerda que en la reunión de planificación del *sprint* se define y planean las funcionalidades a ser desarrolladas en los próximos 30 días. Esta reunión es el momento ideal para analizar y definir el recurso que va a ser el cuello de botella del *sprint* y los miembros de equipo deben tomar en cuenta las restricciones de capacidad para su organización.

Con seguridad en el sistema de desarrollo de software el recurso restricción será un desarrollador senior, o grupo de ellos, con habilidades específicas y críticas en el

[28] En la industria de desarrollo de software el recurso restricción es generalmente un desarrollador o un grupo de ellos. El termino *explotar* usado por la TOC no debe ser mal interpretado y confundido con las malas prácticas de abusos al personal. Explotar el recurso restricción significa aprovechar lo mejor posible ese recurso dentro de las normas legales y éticas.

proceso de desarrollo. Para las empresas resulta bastante difícil conseguir y reclutar esta clase de personal y generalmente son los mejor pagados por las empresas. Por esta razón el tiempo productivo de estos recursos debe ser aprovechado al máximo y esto nos lleva al segundo paso del proceso de mejora.

Para aprovechar la capacidad del desarrollador o desarrolladores que son un recurso restricción se debe proteger que no tengan tiempo ocioso debido a la mala programación de actividades, los cuellos de botella deben tener siempre una lista de actividades por cumplir y definidas por su criticidad. La TOC llama a esta lista de actividades, o inventario por ser procesado, el amortiguador del recurso restricción.

Además estos desarrolladores deben ser protegidos de interrupciones innecesarias y esto se logra implementando estructuras óptimas de comunicación. La metodología Scrum proporciona el marco de trabajo adecuado en este sentido, ya que en los Scrum diarios se define las actividades a ser desarrolladas basándose en la lista de funcionalidades priorizadas por el cliente/usuario y siempre se limitan las reuniones a un tiempo específico para dar preferencia al tiempo invertido en el desarrollo del software.

También siguiendo los lineamientos de Scrum los recursos restricción son potenciados ya que se trata de evitar hacer trabajo innecesario y que no agrega valor para los clientes/usuarios. Las tareas como la creación de reportes o documentación excesiva deben ser eliminadas y en el caso de que decida que son indispensables se debe delegar esas actividades a otros recursos disponibles con capacidad sobrante y no quitar tiempo valioso a los cuellos de botella.

Otras buenas prácticas para explotar la capacidad del recurso restricción son dotarlo de las mejores herramientas para realizar su trabajo y por último, pero de suma importancia, siempre estar seguro de asignar tareas al recurso restricción con requerimientos totalmente claros y validados por los clientes/usuarios.

El tercer paso es subordinar todo el resto de decisiones del negocio a las decisiones tomadas para proteger y explotar al recurso restricción. La subordinación

puede resultar muy difícil de aplicar dependiendo de la cultura de cada empresa. Por ejemplo, podría parecer poco intuitivo dar instrucciones al departamento de ventas de no cerrar nuevos proyectos en los próximos 30 días porque el recurso restricción ya se encuentra utilizado a su máxima capacidad. Otra situación que podría resultar difícil de aceptar es reconocer que los desarrolladores que no son el recurso restricción tengan tiempo ocioso ya que terminaron las actividades asignadas y no se le encargue más trabajo. Sin embargo, tener ocupados al 100% a los recursos no restricción lo único que genera es el incremento de inventario adelante del cuello de botella y posibles distracciones para este recurso, no sólo haciendo que el Throughput no aumente sino poniendo en peligro la generación del mismo.

Todas las decisiones deben tener en cuenta que al recurso restricción sólo se le puede asignar actividades con la misma frecuencia con la que razonablemente pueden ser cumplidas. El resto del sistema debe ser subordinado a este principio. La TOC recomienda que los recursos no restricción se comporten como el “Corre Caminos” de las tiras cómicas, este personaje tiene dos velocidades: a toda velocidad y parado. Los recursos restricción cuando tienen actividades que cumplir las deben hacer a la brevedad posible y luego parar. Este comportamiento minimizará las rupturas en el flujo de desarrollo debido a las fluctuaciones estadísticas en el cumplimiento de actividades y elimina el “síndrome del estudiante”.

El cuarto paso propuesto es elevar la restricción. Es muy frecuente que esto se haga pidiendo al desarrollador o grupo de desarrolladores que son el cuello de botella que trabajen horas extras y generalmente no remuneradas. Esta práctica es muy dañina ya que al largo o mediano plazo los desarrolladores se sentirán exhaustos y su productividad bajará drásticamente poniendo en serio peligro el buen funcionamiento del sistema y la generación de Throughput. El principio 8 del manifiesto Ágil sugiere que los desarrolladores deberían ser capaces de mantener un ritmo constante de forma indefinida y elevar la restricción con horas extras atenta contra este principio, por lo tanto

de debe asignar una carga de trabajo a los recursos que en términos normales puedan afrontar.

El mejor consejo para elevar la restricción es contratar al mejor recurso humano posible para las actividades del cuello de botella o capacitar a los recursos existentes. Este o estos recursos deben ser muy bien pagados y motivados, se debe tener en cuenta que de su rendimiento depende la generación del Throughput total de la empresa.

La empresa debe contar con un plan de contingencia en caso de que el recurso restricción llegue a fallar. Siempre está latente la posibilidad de que los desarrolladores sufran de alguna enfermedad o simplemente dejen la empresa buscando mejores oportunidades. Una alternativa es constantemente capacitar a los desarrolladores junior en las tareas que realiza el recurso restricción. De darse algún imprevisto el o los desarrolladores junior podrían asumir el reto aunque al inicio no sean tan eficientes. También se puede considerar tener candidatos pre seleccionados en el caso de la ausencia definitiva del recurso restricción.

Se cierra el ciclo de mejora con el quinto paso que nos recuerda que no debemos dejar que la inercia se apodere de nosotros. Si para un proyecto o *sprint* se definió el recurso restricción, para el siguiente *sprint* la situación puede cambiar. Cada proyecto de desarrollo de software es único y esto hace que sus requerimientos lo sean también. Como nos sugiere la TOC debemos regresar al paso uno y descubrir el recurso restricción. El proceso de mejora es continuo y se lo debe pulir en cada iteración.

3.3. Planificación de actividades y la Cadena Crítica

En un proyecto de desarrollo de software existen cinco dimensiones que deben ser correctamente administradas para lograr el éxito del proyecto. Estas dimensiones son: el

equipo de desarrollo, el tiempo disponible, las funcionalidades o requisitos, el presupuesto y los recursos disponibles[29].

Todo proyecto cuenta con un grado de incertidumbre y cada una de las cinco dimensiones deben ser protegidas contra la ocurrencia de eventos no planeados. La forma que el sistema puede absorber la incertidumbre es creando amortiguadores para cada una de las dimensiones y cada amortiguador debe ser considerado en las mismas unidades que la dimensión que protege, por ejemplo, el cronograma de un proyecto debe contar con un amortiguador de tiempo que proteja la fecha de entrega; el presupuesto del proyecto debe contar con una cantidad de dinero que ayude a sobrepasar cualquier imprevisto; el equipo de desarrolladores podría ser protegido con personal adicional, pero ya que éste es el principal y más crítico y costoso recurso en la industria del software resultaría casi imposible tener más desarrolladores de los necesarios, por esto la forma lógica de crear un amortiguador es asumir que el tiempo productivo del equipo es de 5 a 6 horas diarias y no las 8 horas laborables; el alcance del proyecto puede ser afectado si se determina la necesidad de funcionalidades críticas no consideradas inicialmente, el amortiguador para estos casos es un listado de funcionalidades no críticas que podrían no ser incluidas en la entrega del proyecto y posiblemente pospuestas para una segunda etapa y así poder incluir los nuevos requerimientos; por último, los recursos de la compañía también deben tener una amortiguador, sería muy irresponsable perder uno o dos días de trabajo por la falla del computador de uno de los desarrolladores.

El tamaño de cada amortiguador dependerá del grado de incertidumbre que se tenga en cada dimensión del proyecto. Cuando la incertidumbre es baja, por ejemplo cuando el proyecto a desarrollarse es muy similar a uno anterior, se recomienda amortiguadores del 15%. Si la incertidumbre es alta el amortiguador deberá ser del 50%

[29] Entre los recursos con los que cuanta la empresa de desarrollo se incluye el espacio físico, hardware, software y aplicativos de oficina, apoyo administrativo.

o más, estos casos se dan cuando se cuenta con desarrolladores nuevos en el equipo o cuando se necesita implementar una nueva tecnología.

Se debe recordar la sugerencia realizada por la TOC de no usar los amortiguadores para proteger las tareas individuales. Los amortiguadores deben ser creados para absorber los eventos no planeados y garantizar el éxito del proyecto como un todo, "siempre se debe recordar que la incertidumbre en el desarrollo de software es inevitable"[30] y se la debe incluir en la planificación de las actividades.

Certeza	Tamaño del amortiguador
100%	15%
90%	25% - 30%
80%	50%
50%- 70%	100%
< 50%	100%

Tabla 4. Certeza del proyecto y tamaño del amortiguador[31]

Como se mencionó, la planificación de las actividades en la metodología de desarrollo Scrum se la hace en la reunión de planificación del *sprint*. En esta reunión el equipo de desarrollo puede apoyarse en los lineamientos de la TOC y definir la Cadena Crítica del *sprint.*

El primer paso será, basándose en la lista de funcionalidades a ser desarrolladas, definir cuáles serán realizadas secuencialmente y cuáles en paralelo en la iteración. Se obtendrá con este ejercicio la ruta crítica del proyecto, se recuerda que la ruta crítica es la secuencia más larga de actividades dependientes a ser realizadas en orden.

[30] David Anderson, *Agile management for software engineering*, Estados Unidos de América, Editorial Prentice Hall PTR, 2004, pág. 47.
[31] Ibíd., pág. 39.

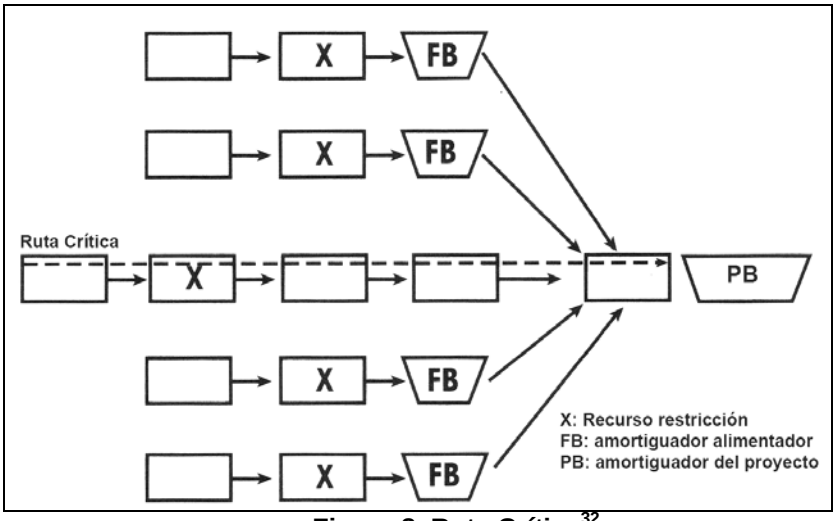

Figura 8. Ruta Crítica[32]

El segundo paso será incorporar la capacidad del recurso restricción en la planificación de actividades. Esta consideración puede cambiar la secuencia de actividades y forzar a actividades que debido a su lógica podrían ser desarrolladas en paralelo ahora tengan que ser desarrollados en forma secuencial ya que son realizadas por el mismo recurso. Se tendrá la Cadena Crítica del proyecto.

El tercer paso será incorporar los amortiguadores necesarios tanto a las rutas de alimentación para garantizar en flujo de la Cadena Crítica como el amortiguador del proyecto total. Con el análisis realizado el equipo podrá comprometerse en desarrollar las funcionalidades seleccionadas para el *sprint* y estar seguros de que este compromiso puede ser cumplido realmente.

[32] Eliyahu M.Goldratt, *Critical Chain*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1997, pág. 214

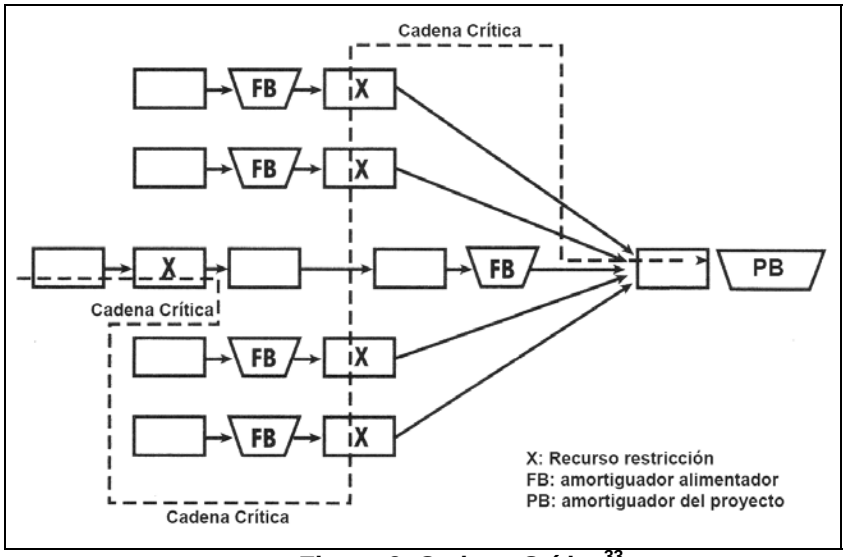

Figura 9. Cadena Crítica[33]

3.4. Estimación de tiempos y administración de amortiguadores

En el apartado anterior se vio la necesidad de proteger la fecha de culminación del proyecto con un amortiguador global y amortiguadores para las rutas que alimentan a la cadena crítica. De esta manera se protege al proyecto y al recurso restricción. Podría parecer que al agregar estos amortiguadores a lo largo al proyecto su duración será mayor; y esto es un punto muy crítico ya que para los clientes el tiempo de entrega es uno de los factores determinantes, al igual que el precio, al momento de seleccionar un proveedor. Se presentarán los lineamientos sugeridos por la TOC para determinar la duración adecuada del proyecto y garantizar su fecha de entrega.

La dificultad para predecir el tiempo que tomará una tarea es muy evidente en los proyectos de desarrollo de software. Generalmente se recurre al desarrollador que está a cargo de la tarea para que proponga un tiempo estimado o los tiempos son consensuados conjuntamente entre el equipo y el jefe del proyecto. Es natural que los tiempos definidos tengan un nivel de confianza bastante elevados, consciente o inconscientemente las personas incorporan un amortiguador para cubrir posibles

[33] Ibid., pág. 218.

imprevistos. Los desarrolladores y el mismo jefe del proyecto no desean incumplir con el cronograma establecido y se presenta el problema de proteger las fechas de las tareas individuales.[34]

En la figura 10 se muestra como al incrementar el nivel de confianza en la definición del tiempo de duración de una tarea se agregan amortiguadores del 100% o más para protegerla en comparación con el promedio histórico o de análisis realizado.

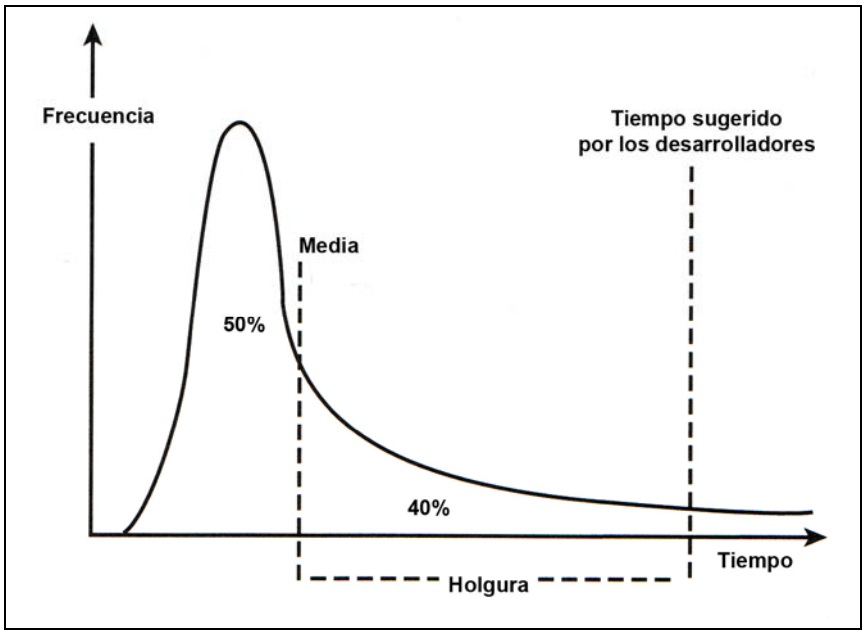

Figura 10. Distribución del tiempo de culminación de actividades[35]

La holgura colocada en los tiempos de cada tarea individual no sirve para proteger la fecha de culminación del proyecto total ya que estos amortiguadores son desperdiciados por dos fenómenos: el síndrome del estudiante y la no acumulación de tiempo ganado.

---

[34] Goldratt irónicamente es su libro Cadena Crítica dice que al momento de hacer un cronograma de actividades 5 más 5 es igual a 13: el jefe de programadores pregunta a dos desarrolladores, cada uno está a cargo de una tarea y las tareas serán realizadas secuencialmente, el tiempo estimado de culminación de cada una. Ellos responden 5 días para cada tarea. El jefe de proyecto agrega estas dos tareas como un solo elemento del proyecto y pone un estimado de 13 días en el gráfico de Gantt.

[35] Eliyahu M.Goldratt, *Critical Chain*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1997, pág 45.

Anteriormente se explico el concepto del síndrome del estudiante, que no sólo es típico de de los estudiantes sino de todos nosotros. Como se sabe que existe holgura de tiempo para cumplir con una tarea no la iniciamos inmediatamente o a un ritmo adecuado. Si un imprevisto o complicación surge en medio camino ya se ha desperdiciado la holgura con la que se contaba y la tarea no será completada a tiempo. El amortiguador que protegía a la tarea local fue desperdiciado.

El segundo problema es que si la tarea es terminada antes de hora, y se cuenta con la holgura ingresada, en muchas ocasiones no es reportado este adelanto en el cronograma. “A un programador no se le ocurriría reportar que acabó antes de hora, él o ella siempre encontrará algo en el código que pueda ser pulido un poco más”[36] Como consecuencia no se acumulan los adelantos, sólo los atrasos. Nuevamente se desperdicia los amortiguadores locales.

La recomendación realizada por la TOC es utilizar el tiempo que duraría cada tarea sin agregar holguras individuales y reconocer que hay un 50% de probabilidad que cada actividad individual se retrase de la fecha pronosticada. Los desfases individuales son absorbidos por el amortiguador general del proyecto.

El jefe de proyecto o Scrum Master debe monitorear cortantemente el estado del amortiguador del proyecto y de cada uno de los amortiguadores alimentadores. De esta manera se pueden evidenciar posibles problemas y tomar acciones con antelación. En la figura 11 se sugieren los niveles de las alertas según el nivel del amortiguador comparado contra el avance del proyecto o en el caso de un amortiguador alimentador contra el avance de esa ruta.

[36] Eliyahu M. Goldratt, *Critical Chain*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1997, pág 121.

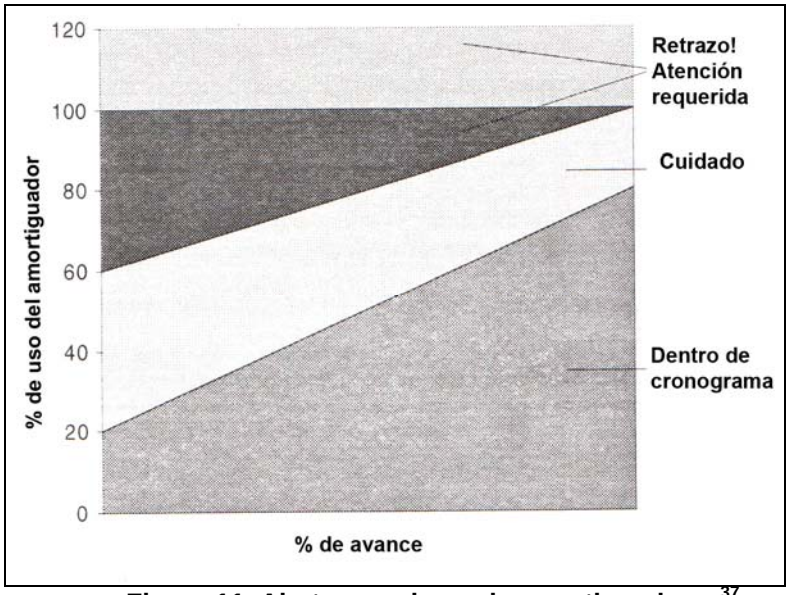

Figura 11. Alertas en el uso de amortiguadores[37]

[37] David Anderson, *Agile management for software engineering*, Estados Unidos de América, Editorial Prentice Hall PTR, 2004, pág. 67.

CAPITULO 4.
INDICADORES DE GESTIÓN EN EL DESARROLLO DE SOFTWARE

4.1. Principios de la Contabilidad de Throughput

La contabilidad de Throughput surgió como una alternativa a la contabilidad de costos tradicional y su principal objetivo es proporcionar información clara, confiable y de una manera ágil para tomar las decisiones que lleven a la compañía a alcanzar su meta siguiendo los principios de la TOC.

La contabilidad de costos no es compatible con la TOC y no proporciona información útil para mejorar el sistema ya que asume que todos los recursos del la compañía son igual de importantes y aboga por la optimización local de todos los centros de costos sin tomar en cuenta si se trata del recurso restricción o no. La optimización local es contradictoria al paso tres del proceso de mejora propuesto por la TOC donde se subordinan todos los otros procesos a la optimización del recurso restricción.

La contabilidad de Throughput tiene un enfoque radicalmente distinto a la contabilidad de costos ya que no toma como válida la premisa de que el costo de un producto depende de la cantidad de recursos de la compañía que se han necesitado para su fabricación ni asigna los costos indirectos ni gastos generales al trabajo en proceso o productos terminados. La contabilidad de Throughput también discrepa con la idea de que al bajar el costo de un producto, calculado con cualquier metodología de asignación, mejorarán las utilidades de la compañía.

La contabilidad de Throughput reconoce que el egreso realizado para contar con un recurso, por ejemplo el sueldo de un obrero, es totalmente independiente de la cantidad de uso de ese recurso. Según el ejemplo propuesto, al obrero se le debe pagar lo mismo así haya estado ocupado las 176 horas laborables del mes o si sólo realizó trabajo productivo por 150 horas. Como se explicó, la TOC no comparte la idea de que todos los recursos deben ser utilizados al 100% para alcanzar el óptimo global, por el

contrario, el tiempo ocioso en los recursos no restricción es necesario para optimizar el sistema como un todo.

En el primer capítulo se presentaron los tres indicadores globales utilizados por la TOC. Se los vuelve a presentar ya que son la base de la contabilidad de Throughput:

- Throughput (T)
- Inversión – Inventarios[38] (I)
- Costos de Operación (OE)

Se define al Throughput como la cantidad de dinero que entra al sistema menos la cantidad pagada a los proveedores. El dinero pagado a los proveedores es principalmente por materia prima o partes que se utilizan en los productos finales y se lo conoce como el Costo Totalmente Variable (TVC[39]), ya que varía directamente con los niveles de producción. Dependiendo de la naturaleza de las operaciones de cada empresa otros costos pueden formar parte de los TVC del producto, pero no hay que confundir este concepto con el utilizado en la contabilidad de costos; sólo cuando verdaderamente el costo varía directamente proporcional al volumen de producción se lo puede calificar como un TVC y su valor debe ser substraído del precio del producto para calcular el Throughput del producto. Otros costos que se incluyen en los TVC pueden ser el desperdicio o mermas, costos de empaque y embalaje y las comisiones de ventas cuando estas representan un valor fijo del precio del producto o servicio.

Para saber el Throughput total generado se debe conocer simplemente el número de productos vendidos, su precio y el TVC, lo que nos lleva a la siguiente fórmula[40]:

[38] Goldratt utiliza la palabra inventario para esta medida, otros autores como Thomas Corbett la llaman inversión. Aunque los términos difieren el concepto es el mismo y se los usará indistintamente.
[39] Se utilizan las siglas en inglés de Totally Variable Cost (TVC)
[40] Thomas Corbett, *Throughput Accounting*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1998, pág. 30.

$T = \sum(P_i – TVC_i) * Q_i$
Tu = P – TVC
Tu: Throughput de cada unidad de producto
P: precio de cada producto
Q: cantidad vendida de cada producto

Tabla 5. Throughput generado

El concepto de Inventario en la contabilidad de Throughput también difiere de la contabilidad tradicional ya que el valor de esté sólo incluye el total de los TVC de los productos. El inventario no absorbe los costos generales de fabricación y ni siquiera los costos directos de mano de obra. El principal objetivo de esta práctica es eliminar la ficticia generación de utilidades por la generación de inventario.

Los activos fijos de la compañía como edificios, muebles, equipos de cómputo, maquinaria también son clasificados como inventario, aunque el objetivo no sea la venta de los mismos y en el caso de que sean vendidos su precio menos el costo de adquisición se trasforma en Throughput.

La última de las tres medidas operativas son los Costos de Operación que es todo el dinero gastado para trasformar el inventario en Throughput. Dentro de los OE están todos los sueldos, incluidos los de mano de obra directa, gastos administrativos, servicios básicos y en general todos los egresos realizados por la compañía para su gestión no contabilizados en los TVC. Cabe recalcar que la contabilidad de Throughput no clasifica los gastos como fijos o variables, lo importante es distinguir entre los que son totalmente variables y los que no.

Las tres medidas de la contabilidad de Throughput nos sirven para tomar decisiones y se convierten en el puente hacia los indicadores de utilidad y rendimiento.

$$U = T - OE$$

$$ROI = U / I$$

U: Utilidad
T: Throughput
OE: Gastos de Operación
I: Inversión
ROI: Retorno sobre la inversión

Tabla 6. Indicadores de resultados[41]

La optimización del sistema se da cuando se explota a la restricción y produce durante todo el tiempo disponible, pero esto no es suficiente, la restricción debe producir siempre los productos o servicios correctos. La contabilidad de Throughput nos ayuda a determinar cuáles son estos productos o servicios que más aportan a la generación de utilidades de la compañía para poder priorizarlos.

El primer paso es reconocer que el tiempo disponible del recurso restricción es limitado y que debe ser aprovechado de la mejor manera, además hay que recordar que el objetivo es maximizar el Throughput generado. Por estas razones se utiliza un indicador que compara el tiempo que demanda el producto o servicio del recurso restricción y el Throughput que este producto o servicio genera. La siguiente tabla ejemplifica mejor este concepto:

Lista de productos

Producto	Precio Neto	TVCs	Throughput ($)	Tiempo en CCR (min)	$/min
A	20.00	8.00	12.00	8.00	1.50
B	25.00	10.00	15.00	15.00	1.00
C	18.00	7.00	11.00	6.00	1.83

Tabla 7. Ejemplo Throughput por producto

[41] Domenico Lepore y Oded Cohen, *Deming and Goldratt The Theory of Constraints and The System of Profound Knowledge*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1999, pág. 26.

En el caso de que la demanda de los productos de la compañía sobrepase a la capacidad instalada, determinada por la capacidad del recurso restricción, se deberá producir toda la demanda del producto C, ya que este es el producto que genera el mayor Throughput por cada minuto de uso de la restricción (ver tabla 7), luego se debe producir el producto A y sólo si sobra todavía capacidad de producirá el producto B.

Para definir el mix de productos óptimo no se ha tomado en cuenta la demanda de cada producto de los recursos no restricción, ya que como se sabe, estos recursos no limitan el Throughput total de la compañía y su utilización es irrelevante para maximizar el Throughput.

Una vez definido el mix de productos a producirse en el período analizado, la compañía puede pronosticar las utilidades potenciales que generará esa decisión. A continuación se presenta un breve ejemplo[42] utilizando la tabla del listado de productos anterior, una tabla de los costos operativos resumida y las formulas previamente analizadas.

Costos mensuales totales de Operación

Rubro	Valor US$
Salarios y beneficios	7,950
Arriendo	800
Servicios básicos	520
Depreciación	1667
Publicidad	800
Otros	120
TOTAL	11,857

Tabla 8. Costos mensuales totales de Operación

[42] Para el ejemplo no se utilizan datos reales de alguna empresa. El objetivo es demostrar la aplicación de las formulas de la contabilidad de Throughput. Para el datos de inversión se tomó arbitrariamente la cifra de $60,000.00

Mix de producción

Producto	Throughput	Demanda	Producción	Utilización CCR	% acum utilización CCR	Throughput por producto
B	15	540	540	4320	43%	8100
A	12	320	320	4800	91%	3840
C	11	420	152	912	100%	1672
				10032[43]		13612

Throughput Total	$ 13,612.00
OE	$ 11,856.67
Utilidad mensual	$ 1,755.33
Utilidad anual	$ 21,064.00
Participación Trabajadores (15%)	$ 3,159.60
Impuesto a la Renta (25)%	$ 4,476.10
Utilidad Neta anual	$ 13,428.30
Inversión	$ 60,000.00
ROI[44]	22%

El otro caso posible es que la demanda de los productos o servicios ofrecidos por la compañía sea inferior a su capacidad instalada, en este caso la restricción es el mercado y se debe producir todo los productos demandados que su precio de venta neto sea superior al total de sus costos totalmente variables. Lo ideal sería poder traer la restricción al interior de la compañía y que no sea ésta el mercado. Para lograrlo se podrían contemplar alternativas para vender la capacidad sobrante produciendo una línea de productos tipo B (productos muy similares con un precio menor de venta) o buscar mercados adicionales. En los dos casos hay que ser muy cauteloso de no canibalizar ni a los productos ni mercados actuales que son los que generan el Throughput principal de la compañía.

[43] Se asume como capacidad máxima mensual del recurso restricción 10,032 minutos. 480 minutos diarios por 22 días laborables en el mes y una eficiencia del 95%

[44] Para el ejemplo se utilizaron OE fijos para el cálculo del ROI, si los OE varían de acuerdo al mix producido estas variaciones deben ser incluidas en el análisis.

El indicador de Throughput sobre tiempo en el CCR nos ayuda a definir el mix de producción óptimo para maximizar el Throughput, pero no se deben olvidar otras variables como consideraciones de marketing, en algunos casos se tendrá que variar el mix de producción para estar acorde a las estrategias de mercado pero siempre sabiendo el costo real que esas decisiones representan.

La contabilidad de Throughput es una herramienta simple y muy poderosa para tomar decisiones que lleven a la compañía a su meta de ganar dinero hoy y en el futuro. Algunas críticas a la contabilidad de Throughput señalan que sólo se enfoca en el corto plazo ya que sólo considera los costos totalmente variables para su análisis y estrategias de precios. Estas apreciaciones son totalmente erradas ya que primero la política de precios de una empresa es fijada tomando en cuenta el mercado y no los costos inherentes a la compañía y segundo la contabilidad de Throughput maximiza la generación de dinero con los que los costos operacionales, sin clasificarlos en fijos o variables, podrán ser cubiertos de la mejor manera posible sin tener que usar criterios de asignación de los costos a los productos individuales, muchas veces de forma subjetiva creando una distorsión para la toma de decisiones.

4.2. Métricas en el desarrollo de software

En el capítulo 2 se definió los tres criterios que debe cumplir un proyecto de software para considerarse exitoso:

i) las características y funcionalidades del producto final

ii) el tiempo de desarrollo

iii) el presupuesto (utilidad esperada)

La persona a cargo de administrar los proyectos de desarrollo de software debe procurar que las tres dimensiones sean cumplidas para que cada proyecto realizado acerque más a la empresa a la meta deseada. En este apartado se resumirán las métricas que ayudarán a monitorear cada una de las dimensiones críticas.

Cumplimiento de características y funcionalidades

La metodología de desarrollo Scrum registra las funcionalidades del producto de software a ser desarrollado por el proyecto en dos documentos: la lista de requerimientos del producto y la lista de requerimientos de cada *sprint*. Estos documentos servirán para comparar lo inicialmente prometido contra lo entregado.

El primer paso que se debe realizar es implementar una unidad que nos permita comparar la cantidad de características ofrecidas con la cantidad entregada. Esta comparación no se la puede realizar directamente al nivel de funcionalidades desarrolladas ya que es evidente que cada funcionalidad demanda un esfuerzo distinto para su desarrollo. Se recomienda calificar a cada funcionalidad por su grado de dificultad con una escala discreta de 1 a 4. Cada unidad de la calificación, llamadas unidades de desarrollo[45], debe representar un determinado número de horas de programación requeridas. Por ejemplo si cada unidad de la escala representa 4 horas de desarrollo y una funcionalidad fue calificada 2, se sabe que ésta requiere 8 horas de análisis, programación y pruebas. No se recomienda definir funcionalidades que sobrepasen la calificación de la escala propuesta, de darse el caso, se necesita subdividir la funcionalidad.

Con esta unidad estándar de tamaño y esfuerzo se podrá determinar de mejor manera los avances realizados y hasta contrastar los resultados entre diferentes proyectos.

Luego del Scrum diario, donde el equipo reporta sus avances, el Scrum Master podrá actualizar al gráfico de Burndown y calcular el siguiente índice de cumplimiento:

[45] El concepto de *unidades de desarrollo* es una adaptación de la unidad de inventario *Puntos de Funcionalidades (story points)* especificada en la metodología de desarrollo Ágil XP. El principal objetivo de implementar una unidad de desarrollo es afrontar el problema de los diferentes "tamaños" de las funcionalidades y su complejidad.

(% de unidades de desarrollo listas / % de días del sprint consumidos)*100

Tabla 9. Indicadores de cumplimiento de funcionalidades

El indicador proporciona el porcentaje de cumplimiento de las funcionalidades prometidas. Al final del sprint el indicador de cumplimiento registrará el éxito del *spint* en la primera dimensión de análisis y al final del proyecto se tendrá el índice general de proyecto.

Otro indicador de suma importancia es la tasa de producción (*R*) del equipo de desarrollo. *R* indica el número de unidades de desarrollo realizadas en un tiempo determinado y permite evidenciar si el equipo de desarrollo está trabajando al ritmo acordado. *R* es la pendiente de la línea que se va formando en el gráfico de Burndown, y se puede utilizar para extrapolar la línea y predecir si todas las funcionalidades podrán ser finalizadas en el *sprint*.

Cumplimiento del tiempo de desarrollo

Para garantizar la fecha de entrega de un *sprint* y del proyecto total se deben seguir los lineamientos ya expuestos de la TOC y la Cadena Crítica. El administrador del proyecto debe monitorear diariamente los amortiguadores de alimentación y el amortiguador del proyecto. Los indicadores de uso de cada amortiguador darán alertas tempranas para tomar acción y prevenir el retraso del proyecto.

(días consumidos del amortiguador / días en el amortiguador)*100

Tabla 10. Indicadores de consumo del los amortiguadores

Por ejemplo, una ruta de alimentación tiene un tiempo de duración de 10 días, un amortiguador de 3 días y estando en el día 8 la penetración en el amortiguador es de 2 días. El amortiguador se encuentra utilizado el 66% y el avance de la ruta es del 60% (faltan 4 días de trabajo estimado), comparando estos datos en el grafico de alertas de

los amortiguadores (figura 11) el administrador de proyecto sabe que debe enfocar su atención en las tareas de esa ruta para evitar un retraso que afecte a la Cadena Crítica y al proyecto total.

Cuando se calcula el porcentaje de utilización del amortiguador del proyecto y se lo compara con el porcentaje de avance se debe recordar que el avance del proyecto está únicamente dado por el avance realizado en la cadena crítica. No se toman en cuenta los trabajos realizados en las rutas de alimentación para calcular el porcentaje ejecutado del proyecto. Este criterio está acorde con el principio de que la cadena crítica define el tiempo necesario para el proyecto.

Al finalizar el proyecto se debe calcular el indicador de cumplimiento del plazo estipulado. El plazo del proyecto incluye el tiempo que se definió para el amortiguador del proyecto.

(días estimados – días reales / días estimados)*100

Tabla 11. Indicadores de desfase de plazo

Este indicador proporciona el porcentaje de desfase en el plazo pactado del proyecto. Si el indicador tiene signo positivo el proyecto terminó antes de la fecha de entrega y el signo negativo evidencia un retraso.

Cumplimiento del presupuesto y rentabilidad

En el apartado anterior se presentaron los indicadores operativos y de resultados propuestos por la TOC. Estos indicadores servirán para validar la rentabilidad del proyecto.

Tp = P – TVC Tp: Throughput del proyecto P: precio cobrado por el proyecto TVC: Costos totalmente variables del proyecto

Tabla 12. Throughput del proyecto

U = T – OE ROI = U / I U: Utilidad antes de impuestos y participaciones T: Throughput OE: Gastos de Operación I: Inversión ROI: Retorno sobre la inversión

Tabla 13. Indicadores de resultados del período

Para calcular el Throughput del proyecto se restra del precio cobrado los TVC, y cabe recalcar que estos costos son únicamente los costos adicionales en los que incurrió la empresa por la existencia del proyecto, por ejemplo la comisión del vendedor, costos legales pagados para la elaboración del contrato, costos de la pólizas de seguro, movilización y viáticos.

Los costos adicionales, como los sueldos de los desarrolladores, deben ser incluidos en los costos de operación (OE). Se recomienda incluir en los OE únicamente los costos directos y no buscar mecanismos de asignación para el resto de costos, los cuales serán tomados en cuenta al calcular la rentabilidad global de la compañía y no a nivel de proyecto.

El último indicador que debe ser monitoreado es la cantidad de inventario en unidades de desarrollo promedio que mantienen la compañía. Este indicador está directamente relacionado con la liquidez de la empresa, se debe recordar que se genera Throughput únicamente cuando el cliente haya pagado por las funcionalidades

entregadas. Si las funcionalidades de un *sprint* ya fueron entregadas y están siendo usadas por el cliente pero estas continúan como inventario si no se ha recibido el pago respectivo. Se recuerda que el objetivo de una empresa de software no es únicamente desarrollar buen código a aplicativos, se debe obtener una utilidad ya que la meta acordada es generar dinero hoy en el futuro para garantizar la supervivencia de la compañía. Un mal manejo de la liquidez de puede llevar a la quiebra a la empresa pese a reportar utilidades ya que no podrá afrontar el pago de sus costos de operación.

Este indicador se lo puede expresar en días de inventario utilizando la siguiente fórmula:

Cantidad de inventario actual / promedio de unidades de desarrollo por día

Tabla 14. Indicador días de inventario

Se recuerda que en la reunión de revisión del *sprint* se debe obtener la aprobación del cliente de las funcionalidades desarrolladas, o en el peor de los casos un par de días después luego de incorporar las observaciones realizadas. Si se utilizan *sprints* de 30 días un número adecuado de días de inventarios estará cerca a 45 días. Se debe considerar la liquidez de la empresa al momento de acordar los términos de pago con el cliente y hacer una adecuada y constante gestión de recuperación de cartera.

Generalmente se acepta una tolerancia del 5% para los indicadores de cumplimiento propuestos en este apartado. Conforme el quipo de desarrollo vaya adquiriendo más experiencia se podrá ir disminuyendo el porcentaje de desfase aceptado para impulsar un proceso de mejora continua.

CAPITULO 5.
CASO PRÁCTICO

5.1. Inicio del proyecto

Se utilizara en esta tesis un proyecto real para ilustrar la aplicación de los conceptos presentados en los capítulos anteriores y se analizara a detalle el primer *sprint* del mismo. El proyecto presentado tenía como objetivo desarrollar un conjunto de Dashboards[46] con información útil para inversionistas locales y extranjeros sobre la actividad industrial en el país, niveles de exportaciones e importaciones, oferta y demanda organizada por familias de productos y sectores productivos. El producto final a entregarse fue nombrado Sistema de Inteligencia de Mercados.

Este proyecto, como en general todos los proyectos de desarrollo de software, inició con una reunión entre el cliente y el dueño del producto para definir los requerimientos del proyecto a ser desarrollado. El cliente es el responsable de especificar en esta reunión sus necesidades a un nivel macro. Conjuntamente se crea un documento llamado Requerimientos del Producto que es un resumen corto que describe el proyecto. Este documento servirá de base para que el equipo de desarrollo lo analice y defina las funcionalidades necesarias a implementarse, la arquitectura de la solución propuesta, el número de iteraciones y las entregas parciales programadas.

[46] La traducción de dashboard al español es panel de control. En el área de gestión empresarial se conoce como dashboard a un conjunto de indicadores y gráficos que analizan un área del negocio.

Requerimientos del producto

<table>
<tr><td>Nombre del Producto:</td><td>Sistema de Inteligencia de Mercados</td></tr>
<tr><td>Fecha de inicio:</td><td>1 de dic 2009</td></tr>
<tr><td>Fecha de entrega:</td><td>29 de enero20 10</td></tr>
<tr><td>Visión del Producto:</td><td>Desarrollar un conjunto de Dashboards con tecnología 100% Web para proporcionar al visitante información de gran valor sobre comercio exterior y las diferentes industrias en el Ecuador</td></tr>
<tr><td>Equipo Asignado:</td><td>Programadores: J.Tenorio, A. Tixilema, Diseño gráfico: E. Proaño</td></tr>
<tr><td>Dueño de Producto:</td><td>J. Prado</td></tr>
<tr><td>Scrum Master:</td><td>N. Palacios</td></tr>
<tr><td>Cliente/Dueño de Producto:</td><td>xxxx</td></tr>
<tr><td>Observaciones:</td><td>Es un producto nuevo a ser desarrollado. Se desarrollará una interface frontal y una interface administrativa multi-usuario y multi-nivel con módulos para la actualización del sistema.</td></tr>
<tr><td>Arquitectura:</td><td>Apache 2 Web Server, PHP 5.2, PostgreSQL 8.3, Corda PopChart 7.2</td></tr>
<tr><td>Listado de Características:</td><td>Front End
- Dashboards
Importaciones
Exportaciones
Balanza Comercial por producto
Balanza Comercial por País
Producción Industrial
Entorno Económico
- Buscador de documentos
- Vista impresión
- Exportar a Excel
- Exportar a PDF
- Page Flip de Documentos
-Traducción a Ingles
Back End
- Usuarios
- Importadores de Excel para cada Dashboard
- Importador de documentos</td></tr>
<tr><td>Plan de entregas:</td><td>3 iteraciones de 3 semanas cada una
<table>
<tr><th>1. 21 Dic 09</th><th>2. 11 Ene 10</th><th>3. 29 Ene 10</th></tr>
<tr><td>Front End
Importaciones
Exportaciones
Producción Indust.
Back End
Usuarios
Importadores Excel</td><td>Front End
Entorno Económico
B. C. por producto
B.C. por país
Documentos
Back End
Importadores Excel</td><td>Front End
Vista impresión
Exportar Excel
Exportar PDF
Page Flip
Traducción
Documentación
Entrega</td></tr>
</table></td></tr>
</table>

El quipo de desarrollo el Scrum Master y el dueño del producto dejaron claros los siguientes puntos:

- La unidad de desarrollo: 4 horas de análisis, programación, pruebas o instalación
- El porcentaje de los amortiguadores es: 20%
- La velocidad del equipo: 3.2 unidades diarias[47] y se tienen 6.4 horas productivas al día
- Días para desarrollo en el sprint: 13
- Unidades potenciales: 42 unidades de desarrollo
- El área de desarrollo, los programadores AT y JT, es el recursos restricción

5.2. Planificación del sprint

Luego de que el equipo haya analizado los requerimientos del proyecto se realiza la reunión de planificación del primer *sprint*. En esta reunión debe asistir el dueño del producto por el lado del cliente, el dueño del producto de la empresa desarrolladora, el Scrum Master y el quipo de desarrollo. Se estudian las características del producto y se las detalla a nivel de funcionalidades y se contrasta con la capacidad del equipo asignado. En base a ese análisis el equipo crea la lista de funcionalidades del *sprint* y se compromete con la fecha y funcionalidades a ser entregadas en la iteración. La lista de funcionalidades del *sprint* sirve para crear el cronograma, definir el recurso restricción y definir los amortiguadores necesarios.

En el primer *sprint* el programador AT fue asignado el mayor número de unidades de desarrollo, un total de 20. Ya que sus labores representaron la cadena crítica del proyecto se definió un amortiguador del proyecto de 4 unidades. Fueron definidos amortiguadores de 1 unidad para las tareas de diseño gráfico y un amortiguador de 2

[47] Se divide las 16 horas diarias disponibles de los 2 desarrolladores para las 4 horas de cada unidad de desarrollo y se aplica el amortiguador del 20%. (1.6 unidades por desarrollador)

unidades para el programador JT, ya que si el amortiguador era mayor sus actividades serían la cadena crítica. JT podía con esa holgura ayudar en las actividades de AT en caso de requerirse.

A continuación el documento resultante de la reunión de planificación del *sprint*:

Lista de funcionalidades del sprint

Tarea	Tipo	Unidades	Responsable
Diseño Gráfico Home	D	1	EP
Diseño Gráfico Internas	D	1	EP
Diseño Gráfico Back End	D	1	EP
Tablas BD Exportaciones/Importaciones	BD	1	AT
Plantillas Corda Exportaciones/Importaciones	P	1	JT
Carga datos Exportaciones	P	2	AT
Admin Usuarios	P	1	JT
Admin Exportaciones	P	4	JT
Mark up y CSS	P	2	AT
Dashboard Exportaciones	P	4	AT
Test & Deploy Exportaciones	TD	1	AT
Carga datos Importaciones	P	2	JT
Admin Importaciones	P	2	JT
Dashboard Importaciones	P	3	AT
Test & Deploy Importaciones	TD	1	AT
Test & Deploy Admin Export/Import	TD	1	JT
Tablas BD Industrias	BD	1	JT
Plantillas Corda Industrias	P	1	AT
Carga datos Industrias	P	2	JT
Admin Industrias	P	3	JT
Dashboard Industrias	P	3	AT
Test & Deploy Industrias/Admin	TD	1	AT
Aseguramiento de Calidad	QA	2	AT/JT
TOTAL		**41**	

TIPO	Total
D	3
P	30
BD	2
TD	4
QA	2
TOTAL	**41**

Responsable	Total
EP	3
AT	20
JT	18
TOTAL	**41**

D: Diseño gráfico
P: Programación
BD: Base de Datos
TD: Test & Deploy (Pruebas e instalación)
QA: Aseguramiento de Calidad

Cronograma del sprint

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Sprint 1**	**16.62 días**	**mié 12/2/09**	**lun 12/21/09**
2	Diseño Grafico Internas	4 horas	mié 12/2/09	mié 12/2/09
3	Diseño Grafico Back End	4 horas	mié 12/2/09	mié 12/2/09
4	Amortiguador de EP 1	3.9 horas	jue 12/3/09	jue 12/3/09
5	Diseño Grafico Home	4 horas	lun 12/7/09	lun 12/7/09
6	Amortiguador de EP 2	4 horas	lun 12/7/09	lun 12/7/09
7	Tablas BD Exportaciones/Importaciones	4 horas	mié 12/2/09	mié 12/2/09
8	Plantillas Corda Exportaciones/Importaciones	4 horas	mié 12/2/09	mié 12/2/09
9	Carga datos Exportaciones	8 horas	mié 12/2/09	jue 12/3/09
10	Admin Usuarios	4 horas	mié 12/2/09	mié 12/2/09
11	Admin Exportaciones	12 horas	jue 12/3/09	vie 12/4/09
12	Mark up y CSS	8 horas	jue 12/3/09	vie 12/4/09
13	Dashboard Exportaciones	16 horas	vie 12/4/09	mar 12/8/09
14	Test & Deploy Exportaciones	4 horas	mar 12/8/09	mar 12/8/09
15	Carga datos Importaciones	8 horas	vie 12/4/09	lun 12/7/09
16	Admin Importaciones	8 horas	lun 12/7/09	mar 12/8/09
17	Dashboard Importaciones	12 horas	mié 12/9/09	jue 12/10/09
18	Test & Deploy Importaciones	4 horas	jue 12/10/09	jue 12/10/09
19	Test & Deploy Admin Export/Import	4 horas	mar 12/8/09	mar 12/8/09
20	Tablas BD Industrias	4 horas	mié 12/9/09	mié 12/9/09
21	Plantillas Corda Industrias	4 horas	vie 12/11/09	vie 12/11/09
22	Carga datos Industrias	8 horas	mié 12/9/09	jue 12/10/09
23	Admin Industrias	12 horas	jue 12/10/09	vie 12/11/09
24	Amortiguador JT	7.9 horas	lun 12/14/09	lun 12/14/09
25	Dashboard Industrias	12 horas	vie 12/11/09	lun 12/14/09
26	Test & Deploy Industrias/Admin	4 horas	mar 12/15/09	mar 12/15/09
27	Aseguramiento de Calidad	8 horas	mar 12/15/09	mié 12/16/09
28	Amortiguador del Proyecto	16 horas	mié 12/16/09	vie 12/18/09
29	Cierre de Sprint	8 horas	vie 12/18/09	lun 12/21/09

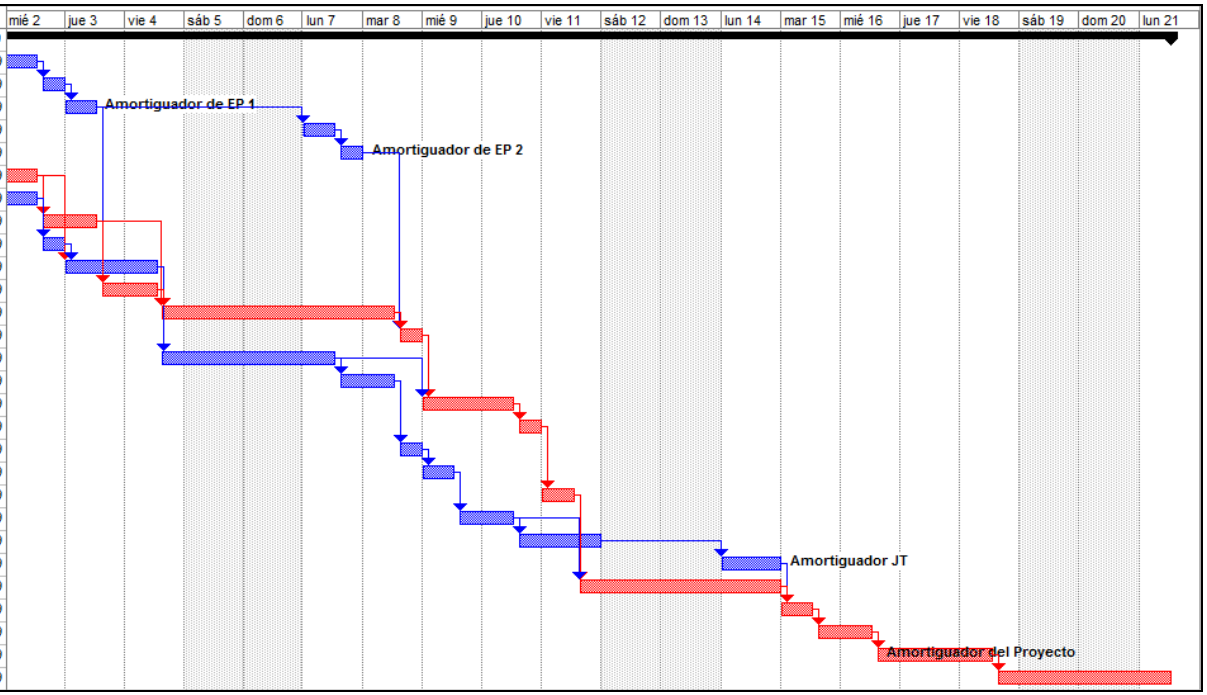

Figura 12. Cronograma sprint

5.3. Seguimiento del sprint

El Scrum Master se reunió con el equipo todos los días de 9:05 a 9:20 de la mañana para realizar el Scrum Diario. Luego de cada reunión de seguimiento se actualizaba la lista de funcionalidades del *sprint* y se monitoreaba el estado de los amortiguadores. Hubo imprevistos en la construcción de los tres dashboards planificados, los desfases causados fueron absorbidos por el amortiguador del proyecto. Gracias la holgura que se tenía con el programador JT se pudo superar el desfase en la cadena crítica que se evidencio al finalizar el *sprint*. El día 14 del sprint[48] se realizó la presentación de las funcionalidades desarrolladas, probadas e instaladas en el servidor de pruebas. En esta reunión de cierre y revisión del *sprint* el cliente realizó observaciones y ajustes menores a las funcionalidades desarrolladas. Se acordó realizar estos cambios y mejoras de inmediato y posponer por un día el inicio de la siguiente iteración.

A continuación los reportes de Scrum que resumen la ejecución del *sprint*:

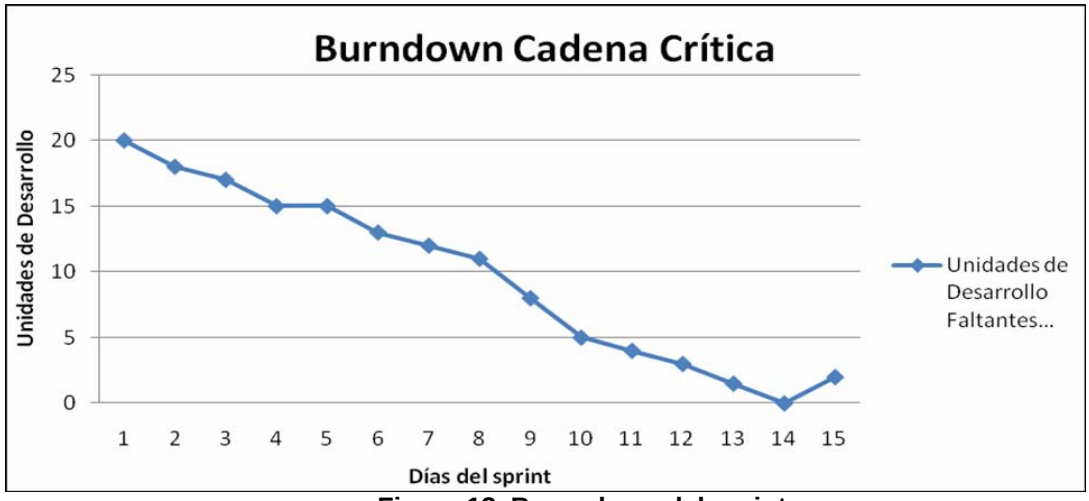

Figura 13. Bourndown del sprint

[48] En este ejemplo se usaron días laborables para el sprint para poder llevar un mejor control y claridad

Tarea	Tipo	Responsable	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Diseño Grafico Home	D	EP	1														
Diseño Grafico Internas	D	EP	1	1													
Diseño Grafico Back End	D	EP	1	1	1												
Tablas BD Export/Importaciones	BD	AT	1														
Plantillas Corda Export/Importaciones	P	JT	1														
Carga datos Exportaciones	P	AT	2	1													
Admin Usuarios	P	JT	1	1													
Admin Exportaciones	P	JT	4	4	4	2											
Mark up y CSS	P	AT	2	2	2												
Dashboard Exportaciones	P	AT	4	4	4	4	4	2	1								1
Test & Deploy Exportaciones	TD	AT	1	1	1	1	1	1	1	1							
Carga datos Importaciones	P	JT	2	2	2	2	2	2									
Admin Importaciones	P	JT	2	2	2	2	2	2	1								
Dashboard Importaciones	P	AT	3	3	3	3	3	3	3	3	1						1
Test & Deploy Importaciones	TD	AT	1	1	1	1	1	1	1	1	1						
Test & Deploy Admin Export/Import	TD	JT	1	1	1	1	1	1	1	1							
Tablas BD Industrias	BD	JT	1	1	1	1	1	1	1	1							
Plantillas Corda Industrias	P	AT	1	1	1	1	1	1	1	1	1	1					
Carga datos Industrias	P	JT	2	2	2	2	2	2	2	2	2	1					
Admin Industrias	P	JT	3	3	3	3	3	3	3	3	3	3	2	1			
Dashboard Industrias	P	AT	3	3	3	3	3	3	3	3	3	2	2	1			1
Test & Deploy Industrias/Admin	TD	AT	1	1	1	1	1	1	1	1	1	1	1	1	1		1
Aseguramiento de Calidad	QA	AT/JT	2	2	2	2	2	2	2	2	2	2	2	2	1		
TOTAL			**41**	**37**	**34**	**29**	**27**	**25**	**21**	**19**	**14**	**10**	**7**	**5**	**2**		**4**
Cadena Critica			**20**	**18**	**17**	**15**	**15**	**13**	**12**	**11**	**8**	**5**	**4**	**3**	**1.5**	**0**	**2**

Amortiguadores															
EP 1	0%	0%													
EP 2	0%	0%	100%												
JT	0%	30%	60%	40%	20%	100%	30%	60%	80%						
Proyecto	0%	0%	25%	25%	75%	75%	100%	125%	100%	75%	100%	150%	100%	100%	150%

Indicadores del Sprint

- Cumplimiento de funcionalidades: 100% todas las funcionalidades fueron desarrolladas
- Tasa de Producción R: 38 udd / 13 días = 2.9 udd/día
- Desfase de plazo: (1 día/ 13 días)x100 = 7.7%

El proyecto analizado tenía sólo dos rutas principales de actividades, cada una asignada a un desarrollador, por lo que no demandaba un alto grado de complejidad para coordinar los trabajos entre los involucrados. Pese a esto, la implementación de Scrum fue de gran utilidad para organizar las actividades desde el primer día de trabajo y poder mantener un control diario de los avances en el proyecto.

La costumbre del Scrum diario permitió al equipo discutir y analizar problemas presentados en el desarrollo de ciertas funcionalidades y en conjunto como equipo llegar a la mejor manera de implementar los requerimientos. Scrum incentiva la colaboración entre los miembros con lo que se logra mejor productividad de todos los desarrolladores.

El sistema de políticas amortiguadores de la TOC fue una herramienta de gran ayuda para el administrador del proyecto. Los desfases fueron detectados desde el día de su ocurrencia lo que permitió tomar acciones y enfocar los esfuerzos donde más se necesitan. Desde el día 7 del *sprint* se evidenció un potencial retrazo en la Cadena Critica pero este fue evitado gracias a su detección temprana. El uso de amortiguadores permitió optimizar los tiempos de las tareas ya que se eliminó la holgura localizada, y la holgura global asignada por la incertidumbre absorbió las fluctuaciones inevitables.

Los reportes de Scrum permitirán analizar el trabajo realizado y con mejor juicio definir el número de unidades de desarrollo establecidas para cada funcionalidad y el tamaño de los amortiguadores en futuras iteraciones o proyectos.

CONCLUSIONES Y RECOMENDACIONES

En esta tesis han sido expuestos los criterios de la TOC y cómo estos pueden ser utilizados para optimizar el proceso de desarrollo de software. La norma en los proyectos ha sido su alto grado de incertidumbre y como consecuencia los atrasos frecuentes en las fechas de entrega. Entendiendo que el proceso de desarrollo es un sistema con una restricción que lo limita se pueden aplicar los 5 pasos propuestos por Goldratt para optimizarlo. Además, los argumentos lógicos de la Cadena Crítica son una herramienta muy poderosa para afrontar la incertidumbre y a pesar de que ésta exista se pueden terminar los proyectos dentro del tiempo estipulado.

También se han presentado los fundamentos de la metodología de desarrollo Ágil Scrum y como este marco de trabajo proporciona la disciplina necesaria para organizar las actividades. El control sobre las tareas se facilita al dividir todo el proyecto en varias fases o *sprints* y dentro de cada *sprint* tener muy bien definidas las actividades a realizarse, su grado de complejidad y tiempo estimado de desarrollo. Las varias reuniones periódicas propuestas por la metodología, lejos de ser burocráticas, permiten tener un alto nivel de comunicación entre los involucrados y mantener la información del proyecto visible permanentemente para mejorar el proceso de toma de decisiones.

Se ha demostrado que se puede conjugar los lineamientos de la TOC y los principios de Scrum para mitigar los problemas frecuentes en los proyectos de software: requerimientos poco claros y cambiantes, la falta de involucramiento de las partes intervinientes, la ineficiencia en el uso de los recursos disponible y la mala administración financiera. Con la metodología de la Cadena Crítica se evidencia la restricción de cada proyecto, se mitiga la incertidumbre con la correcta asignación y administración de los amortiguadores, lo que proporciona alertas tempranas para los desfases en el cronograma y la rigurosidad proporcionada por Scrum permite al equipo estar siempre enfocado y lograr cumplir su compromiso.

Por último se han analizado las convenciones de la contabilidad de Throughput para definir indicadores de operación y de resultados que determinen el rumbo para alcanzar la meta de una empresa de desarrollo de software: Generar dinero hoy en el futuro, ejecutando proyectos exitosos que cubren las expectativas del cliente, de una manera rentable y cumpliendo el cronograma establecido.

Una recomendación general para implementar Scrum y TOC es primero familiarizar al equipo de desarrollo con los principios y valores propuestos por la metodología Ágil. Luego de que el Scrum Master y los desarrolladores se hayan familiarizado con la dinámica de Scrum se podrá optimizar el sistema implementando los lineamientos de la TOC para crear un proceso de mejora continua y la aplicación del concepto de la cadena Crítica para definir el cronograma del proyecto y amortiguadores.

Aunque haya inicialmente fallas en la implementación rigurosa de Scrum y TOC, los resultados de aplicar la metodología propuesta serán evidentes al mejorar el cumplimiento de la fecha de entrega de los proyectos, la optimización del uso de los recursos y la rentabilidad conseguida.

El equipo deberá profundizar constantemente sus conocimientos en el uso de las herramientas expuestas para que con cada proyecto se encamine a la empresa hacia su meta.

GLOSARIO

Ágil: Una manera iterativa e incremental de desarrollo de software que sigue el Manifiesto Ágil y sus principios relacionados.

Burndown, gráfico: Es un gráfico que muestra la cantidad de trabajo restante en una iteración. El eje X indica el tiempo y el eje Y la cantidad de trabajo.

Equipo de desarrollo: Un grupo interdisciplinario de profesionales encargado del desarrollo del software.

Gastos de Operación (OE): Todo el dinero gastado en convertir el inventario en Throughput. En el desarrollo de software es el dinero gastado para convertir los requerimientos de los clientes/usuarios en software funcional.

Indicadores de resultados: Miden el desempeño financiero de la organización.

Indicadores operativos: Miden la eficiencia y el desempeño de las operaciones o procesos dentro de la organización.

Inventario (I): Todo el dinero invertido por el sistema en cosas a ser vendidas. En el sistema de desarrollo de software se entiende por las funcionalidades no cobradas al cliente y los costos totalmente variables en los que se ha incurrido.

Iteración: Un periodo de tiempo determinado durante el cual el equipo planea, desarrolla, prueba y entrega un conjunto de funcionalidades.

Manifiesto Ágil: Declaración de principios y valores que definen el desarrollo ágil de software

Meta: El objetivo para el cual el sistema fue creado.

Scrum diario: Una reunión corta al iniciar el día de trabajo donde el equipo sincroniza sus actividades y los avances realizados.

Scrum Master: persona responsable del proceso de Scrum y guía del equipo de desarrollo.

Scrum: Es una metodología Ágil desarrollada en los años 90 por Jeff Sutherland y Ken Schwaber, propone un desarrollo incremental de un producto por medio de iteraciones.

Sprint: Scrum utiliza el término sprint para referirse a cada iteración del proyecto.

Throughput: Es la tasa a la que se genera efectivo a través de la entrega de software funcional. Sólo cuando el cliente ha realizado el pago por los entregables se ha generado Throughput.

TOC: Las siglas en inglés de Teoría de Restricciones (Theory of Constrains).

Unidades de desarrollo: Definen el nivel de complejidad de una función de desarrollo y de una manera aproximada las horas de trabajo necesarias para concluirla.

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

BIBLIOGRAFÍA

Anderson, David, *Agile management for software engineering*, Estados Unidos de América, Editorial Prentice Hall PTR, 2004.

Corbett, Thomas, *Throughput Accounting*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1998.

Dettmer, H. William, *Goldratt´s Theory of Constraints*, Estados Unidos de América, Editorial ASQC Quality Press, 1997.

Goldratt, Eliyahu M, Et al, *Necesario Más No Suficiente*, México, Ediciones Castillo, 2001.

Goldratt, Eliyahu M., *Critical Chain*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1997.

Goldratt, Eliyahu M., *No fue la Suerte*, México, Ediciones Castillo, 1995.

Goldratt, Eliyahu M., y Fox, Robert E, *La Carrera*, México, Ediciones Castillo, 1999.

Lepore, Domenico y Cohen, Oded, *Deming and Goldratt The Theory of Constraints and The System of Profound Knowledge*, Estados Unidos de América, Editorial The North River Press Publishing Corporation, 1999.

Pressman, Roger S., *Ingeniería del Software, un enfoque práctico*, México, McGraw-Hill, 2006.

Schwaber, Ken, *Agile Project Management with Scrum*, Estados Unidos de América, Microsoft Press, 2004

Sliger, Michele y Broderick, Stacia, *The Software Project Manager's Bridge to Agility*, Estados Unidos de América, Addison-Wesley Professional, 2008