Herramientas de manufactura esbelta aplicadas al desarrollo de software con calidad

Lean manufacturing tools applied to quality software development

Macringer Omaña.[1] M.Sc. & José Tomás Cadenas.[2] M.Sc.
1. Departamento de Tecnología de Servicios, Universidad Simón Bolívar, Venezuela
2. Departamento de Computación y T.I., Universidad Simón Bolívar, Venezuela
3. Investigadores externos Centro de Análisis, Modelado y Tratamiento de Datos, CAMYTD, Facultad de Ciencias y Tecnología, Universidad de Carabobo, Venezuela
macringer@usb.ve, jtcadenas@usb.ve

Recibido para revisión 15 de febrero de 2011, aceptado 28 de junio de 2011, versión final 30 de junio de 2011

***Resumen*— El artículo presenta una propuesta de herramientas de manufactura esbelta aplicadas al desarrollo de software, dentro del marco de métodos ágiles, en particular el Desarrollo de Software Esbelto; lo que permitirá a equipos de desarrollo obtener una calidad sistémica del software (producto, procesos y personas que intervienen). Se recomienda emplear estas herramientas y evaluarlas constantemente con la aplicación iterativa e incremental del ciclo de calidad propuesto por Deming: planificar, hacer, verificar y actuar. Con esta propuesta se contribuye al desarrollo de proyectos de software con calidad en entornos científicos-académicos, ajustados al tiempo planificado y con los recursos presupuestados; utilizando herramientas propias de la Ingeniería.**

***Palabras claves*— Desarrollo de software esbelto; Calidad de software; Método ágil; Ingeniería de software.**

***Abstract*— We presented a proposal of lean manufacturing tools applied to software development within the framework of agile methods, in particular Lean Software Development; this will enable development teams to obtain a systemic quality of the software (product, process and people involved). We recommend using these tools and constantly evaluate with the implementation of iterative and incremental quality cycle proposed by Deming: Plan, Do, Check and Act. This proposal contributes to the development of quality software projects on scientific-academic environments, tight the scheduled time and budgeted resources, using own tools of engineering.**

***Keywords*— Lean software development, Software quality, Agile method, Software engineering**

I. INTRODUCCIÓN

La industria del software, en el ámbito denominado sociedad de la información y el conocimiento, es considerada de gran importancia para la competencia en un mundo globalizado; por esta razón la calidad del desarrollo de software cobra vital importancia.

Las relaciones entre las empresas, gobiernos y personas, ha cambiado por el uso de la tecnología; las empresas están interconectadas continuamente con los clientes y con otras empresas, además, utilizan herramientas de gestión y extracción del conocimiento para ser más eficientes; los gobiernos mejoran su presencia a través la Web y prestan servicios a los ciudadanos; las personas emplean redes sociales para sus relaciones interpersonales. La web 2.0 ha desarrollado una nueva sociedad virtual, donde el eje fundamental es la información [1]; las computadoras y sistemas informáticos están omnipresentes en la sociedad moderna en lo que se denomina computación ubicua.

Desde hace cierto tiempo surgió la rama de Ingeniería del Software, naciendo la inquietud de desarrollar productos de buena calidad, sin que esto signifique un incremento en el uso del tiempo, ni un mayor costo; se sabe que los sistemas informáticos son falibles al igual que la ciencia ya que son elaborados por seres humanos, lo importante es estar siempre en la búsqueda de perfeccionar las técnicas para evitar que los errores de software incidan en la operatividad de una organización o institución [23]. Además, debido al uso generalizado y la confianza de las personas en los sistemas informáticos se hace necesario garantizar que cumplan con las expectativas de calidad y confiabilidad.

El objeto de este artículo es mostrar las posibilidades de desarrollo de software de calidad utilizando herramientas de una filosofía que ha sido exitosa en la elaboración de diversos productos, denominado Manufactura Esbelta o *Lean Manufacturing* [4]. Se revisaron los antecedentes que existen y se instrumenta, mediante herramientas específicas para un equipo de desarrollo de software en un entorno científico-académico, la aplicación de dicha filosofía, considerada un método de desarrollo ágil por diversos autores [2][6].

A continuación se presentan secciones donde se explica el desarrollo de software esbelto, los antecedentes de esta

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

investigación, la calidad del software, las herramientas de desarrollo esbelto propuestas, finalizando con las conclusiones y recomendaciones.

II. DESARROLLO DE SOFTWARE ESBELTO

La empresa Toyota revoluciona en la década de los 80 a la industria automotriz con su sistema de producción que promueve la eliminación del desperdicio [10], resalta la cadena de valor del producto, manufactura bajo demanda (utilizando técnicas como la de justo a tiempo) y se enfoca en la gente que agrega valor. Womack, Jones y Roos [28] son los primeros autores que adoptan el termino esbelto (*lean*) para referirse a esta filosofía de trabajo como un tipo de pensamiento denominado posteriormente *Lean Thinking* [27].

La prioridad del sistema de producción de Toyota (TPS) es la velocidad entendida como: la capacidad de satisfacer la demanda del mercado con un sistema de producción en sintonía con la necesidad; y perfección: sólo siendo perfecto se puede ser rápido, sólo si no hay desperdicio (*muda* en japonés) se puede adquirir la velocidad necesaria.

Iniciativas esbeltas en manufactura, logística, servicios y desarrollo de productos han permitido mejoras dramáticas en costo, calidad y tiempo de entrega. Estos beneficios pueden obtenerse trasladando ciertas técnicas aplicables en el desarrollo de software [15].

El pensamiento esbelto capitaliza la inteligencia de las personas que están más cerca de la agregación de valor al producto, ya que tienen la convicción de que son ellos quienes determinan y mejoran continuamente la manera de ejecutar su trabajo, tomando decisiones con autonomía; esto es a lo que se llama empoderar al equipo de trabajo [14].

Las técnicas de calidad aplicadas a la industria manufacturera no son trasladables automáticamente a la industria del software, sin embargo Poppendieck y Poppendieck [16] han conceptualizado el desarrollo de software esbelto (*Lean Software Development*), el cual se basa en siete principios: eliminar el desperdicio, construir incrementalmente el desarrollo con calidad (mejoramiento continuo), preservar el conocimiento, diferir compromisos (tomar decisiones en el último momento responsable), entregas rápidas de funcionalidades (comprobadas), dar autonomía a las personas (empoderar el equipo de trabajo) y optimizar la visión completa (perspectiva sistémica del proyecto).

Nociones tales como el desperdicio en manufactura se asocian a: generación de códigos inconclusos, sobre-documentación (papeleo excesivo), defectos (*bugs*), re-trabajos o asignar una persona a múltiples proyectos; evitar el transporte de productos es asemejado al cambio de actividades antes de finalizar funcionalidades; la sobre-producción es relacionada con la generación de características no requeridas por los usuarios finales.

Poppendieck y Poppendieck [17] recomiendan la implementación de seis disciplinas: organizar el área de trabajo, establecer estándares (codificación, denominación, interfaz gráfica), control de versiones, procesos de construcción, integración frecuente y fijar políticas de pruebas (alcance, frecuencia, automatización y niveles de defectos).

Además, las leyes de manufactura esbelta, según los citados autores, pueden aplicarse en el desarrollo de software en la siguiente forma: los clientes necesitan definir el nivel de calidad (pueden cambiar sus mentes), la velocidad es proporcional al tiempo desperdiciado en colas y esperas, el 20% de las actividades produce el 80% de los retrasos, la velocidad es inversamente proporcional a trabajos parcialmente elaborados (es preferible realizar implementación de conjuntos de características pequeñas en ciclos cortos de liberaciones de productos) y los costos de la complejidad son mayores que los de defectos o retrasos (no añadir características extras).

Asimismo afirman que la velocidad, calidad y el costo son características de un producto de software que están inexorablemente ligadas y todas pueden mejorarse sin ser afectadas mutuamente. Es por ello que la madurez en el desarrollo de software puede medirse por la velocidad en la cual un requerimiento del usuario puede ser trasladado al software repetidamente en forma confiable.

En términos generales se puede decir que el desarrollo esbelto logra expandir los basamentos teóricos del método ágil [8] aplicando los principios bien conocidos y aceptados de manufactura esbelta a la elaboración de software. Pero va más allá proporcionando herramientas de gestión para ayudar a trasladar los principios esbeltos a las prácticas ágiles que son apropiadas para dominios individuales.

III. ANTECEDENTES

Es importante destacar el artículo de Omaña y Cadenas [11], quienes hicieron una investigación de campo, apoyados en una revisión documental de tipo no experimental, donde evaluaron un producto de software (SQLfi V4) desarrollado por un grupo de proyecto en un entorno científico-académico, utilizando el Modelo Sistémico de Calidad (MOSCA), obteniendo un nivel de calidad sistémico nulo. En base a ello, proponen la adopción de un modelo de desarrollo para la construcción de software de calidad basado en estándares establecidos de manufactura esbelta, con los aportes derivados de la evaluación y la experiencia de los investigadores en áreas de Ingeniería Industrial y desarrollo de software.

Es de hacer notar que en la presente propuesta se instrumenta dicho modelo a través de la utilización de herramientas específicas de desarrollo de software esbelto, con el propósito de lograr un mayor grado de madurez en equipos de proyectos en entornos científicos-académicos.

Por otra parte en [22] los autores presentan un artículo donde indican que las compañías se enfrentan a serios compromisos para entregar productos de software personalizados en períodos cortos de tiempo, afirman que esta situación no es bien gestionada por metodologías tradicionales, por lo que se recomiendan las ágiles para proyectos pequeños o medianos (*Scrum* o *eXtreme Programming*); mientras que los principios esbeltos (*Lean*) son aplicados en compañías orientadas al *hardware*. También explican que compañías como Ericsson han desarrollado y aplicado una mezcla de estas dos últimas a lo que denominan Desarrollo Optimizado (*Streamline Development o SD*), la cual promete tiempos cortos para el desarrollo e incrementa la agilidad del mercado cuando se tiene un proyecto de software grande y complejo. El objetivo del artículo es presentar un método sencillo y confiable para predecir el nivel de defectos por resolver en proyectos de desarrollo de software basados en *Lean*.

Por su parte en [13] los autores exponen que la meta del desarrollo esbelto es lograr un flujo continuo y suave de producción de software con máxima flexibilidad y mínimo desperdicio en el proceso. Todas las actividades y productos de trabajo que no contribuyen a agregar valor al cliente son considerados desperdicios, los cuales al ser identificados y eliminados ayudan a centrarse en actividades creativas de valor. En el citado artículo se afirma que es muy difícil lograr la introducción de desarrollo esbelto, porque requiere de un cambio de paradigma sobre los procesos de software, éste no puede ser implementado drásticamente sino de una forma evolutiva e incremental (aplicación del método *kaizén*, por su denominación en japonés). Proponen un método denominado Mejoramiento de Procesos de Software a través de medición esbelta (SPI-LEAM, por sus siglas en inglés). El método permite evaluar el rendimiento de los procesos de desarrollo de software y tomar acciones continuas para lograr alcanzar el desarrollo de software esbelto mejorado en el tiempo.

En [9] se aduce que todavía los proyectos de software son impredecibles, ya que tardan más de lo planificado, exceden el presupuesto asignado o son cancelados antes de ser finalizados; es por ello que el porcentaje de fracasos en estos tipos de proyectos es muy alto. Dichos autores afirman que el análisis ágil, desarrollo de software esbelto, *Scrum* y Programación eXtrema (*XP*) han sido temas de actualidad en los últimos años; por lo que es un gran reto para las corporaciones tomar decisiones inteligentes acerca del cambio en la cultura organizacional que permita obtener proyectos de software completados a tiempo, ajustados al presupuesto y con calidad. Los autores proponen ocho elementos necesarios para el desarrollo de software moderno basado en experiencias exitosas en el área industrial y académica; la meta es mostrar una pauta para minimizar los fracasos en dichos proyectos. En muchos casos los autores presentan buenas prácticas aplicadas en esta área.

Por otro lado, en [5] se asevera que los métodos de desarrollo ágiles tales como XP, Scrum y Desarrollo de Software Esbelto (*Lean SD*), han ganado mucha popularidad durante los últimos años; aunque la idoneidad de estas prácticas en dominios y contextos de negocios diferentes aún no está claro. En su artículo investigan acerca de la aplicabilidad de principios en el contexto de desarrollo de productos de software dirigidos por el mercado (*market-driven*) o MDPD, centrándose en actividades de pre-proyectos. Se presentan resultados de la comparación entre propiedades típicas de métodos ágiles con las necesidades de MDPD, además de los hallazgos de un caso de estudio en la compañía Ericsson (una de las primeras compañías en adoptar el método de desarrollo ágil). Los resultados demuestran que no hay coincidencia entre los principios ágiles y las necesidades de actividades de pre-proyecto en MDPD, por lo que se desaprovechan las ventajas de los métodos ágiles, amenazando el desarrollo a largo plazo del producto por una gestión poco eficiente.

En [3] los autores argumentan que a pesar de existir muchos métodos ágiles, se conoce poco acerca de su puesta en práctica y los efectos que producen. Es por ello que hacen una revisión sistemática para evaluar, sintetizar y presentar hallazgos empíricos sobre desarrollos ágiles a la fecha. Además, proporcionan una visión general de tópicos, fortalezas e implicaciones para la investigación y la práctica. Los principales métodos ágiles revisados en dicho artículo fueron: metodología *Crystal*, método de desarrollo de software dinámico (DSDM), desarrollo dirigido a características, desarrollo de software esbelto, *Scrum*, programación extrema (*XP* y *XP2*). Una de las conclusiones del estudio es que se necesita incrementar la cantidad y calidad de los estudios sobre desarrollo de software ágil; otros métodos distintos a *XP*, tales como *Scrum*, merecen una mayor atención; además, hay una serie de temas de investigación en esta área pendientes por abordarse.

Por otra parte en [25] se afirma que el desarrollo de software esbelto (*Lean SD*) ha recibido mucha atención debido a su nivel de estandarización, el cual es normalmente más alto que lo usual utilizado en otras técnicas similares. Proponen enriquecer el dominio de *Lean SD* para obtener mejores resultados. El objetivo del citado artículo fue contribuir con solucionar problemas en la dirección, planificación y calendarización de actividades durante el proceso de desarrollo de software utilizando *Lean SD*; además, se acentúa el importante rol del conocimiento sobre las acciones necesarias para producir software en plataformas y tecnologías diversas. El enriquecimiento del dominio de *Lean SD* se propone mediante técnicas de planificación de inteligencia artificial (*AI*) e Ingeniería del conocimiento (*KE*).

En [6] se asevera que indicadores de madurez organizacional para el desarrollo de software (niveles CMMI, porcentajes SPICE o estándares ISO) son considerados muy importantes, es así como en las corporaciones existen políticas que obligan a todas las partes de la misma a lograr ciertos niveles. A la vez indican que métodos ágiles (*XP*, *Scrum*, Desarrollo Esbelto y método *Crystal*) siguen ganando renombre, incluso para grandes proyectos de desarrollo de software. Los autores consideran

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

que el artículo fue un punto de partida para lograr ajustes en las metodologías ágiles, que permitan ciertos niveles CMMI de madurez, utilizando un enfoque cualitativo para analizar como los métodos ágiles apoyan o se contraponen a áreas de procesos CMMI. Los autores analizaron XP, generando procedimientos generales acerca de la comparación y compatibilidad entre CMMI y métodos ágiles. Concluyen que extendiendo la visión de los métodos ágiles a una perspectiva más amplia de la organización puede ayudar a utilizar conceptos existentes para la mejora de procesos.

IV. CALIDAD DEL SOFTWARE

En [18] se define la Calidad del Software como la concordancia de los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo documentados y con las características implícitas que se espera de todo producto elaborado profesionalmente; la importancia de cada característica de calidad puede variar dependiendo del tipo de software y del contexto de la organización de desarrollo.

La gestión de calidad del software guarda una estrecha relación con la realizada para los sistemas de manufactura tradicionales al estructurarse básicamente con los mismos procesos: tienen planificación, control y aseguramiento; no sucede así con el mejoramiento, el cual no se señala explícitamente en la gestión del software [20]. Sin embargo, se cree que el mejoramiento está contenido implícitamente en la documentación de los proyectos ya que permite hacer un seguimiento de los resultados obtenidos en proyectos similares con la finalidad de determinar si es necesaria la optimización de los procesos que lo apoyan.

En general las personas esperan que el software cumpla sus funcionalidades en forma correcta (hacer las cosas correctas) y también deben hacer lo que se supone para lo que están desarrollados (hacer correctamente las cosas); esto es sólo uno de los aspectos de la calidad del software.

En [24] se señala que los usuarios confían en computadoras (individuales o interconectadas), así como en la infraestructura global de información y la web, para satisfacer sus necesidades de procesamiento, almacenamiento, búsqueda y recuperación de información. Para ello requieren de software que sea fácil de utilizar (amigable) y confiable.

En este sentido los requerimientos de alta calidad deben ser satisfechos por las personas involucradas en el desarrollo de estos sistemas de software a través de diversas actividades de garantía y deben ser apoyadas por pruebas concretas sobre la base de mediciones.

El término “crisis del software” se ha utilizado para hacer referencia a los problemas que enfrentan las organizaciones de desarrollo debido al aumento de la complejidad de los sistemas; entendiéndose que el éxito de los proyectos implica mucho más que escribir código cumpliendo ciertos requerimientos funcionales. A fin de solucionar esta problemática surgió la disciplina de la Ingeniería del Software donde una de las áreas de trabajo de más auge hoy en día es la calidad.

De acuerdo a lo afirmado en [12] no tiene sentido diseñar un sistema altamente eficiente si no se utiliza, ni diseñar un sistema muy efectivo si no es factible realizarlo por los recursos o el tiempo; por lo tanto, se deben considerar la disponibilidad de personas, tecnología, financiera y tiempo en cualquier diseño para el mínimo cumplimiento de un proceso eficaz, sin la limitación de la efectividad del producto. Si estas relaciones de la calidad no son consideradas en el diseño, la calidad global podría ser pobre.

En [26] se propone un triángulo (producto, tecnología y personas) incluido en el proceso de desarrollo para la certificación de la calidad del software, esto se deduce de la aplicación del principio del enfoque de sistemas, ya que la naturaleza de los sistemas no puede ser dividida en partes, sino que debe existir una interdependencia y colaboración entre las partes para que el mismo sea visto como un todo (ver figura 1).

Figura 1. Determinantes de la calidad del software

Un producto es un bien tangible resultado de un proceso, el software incluye documentos asociados; la estandarización del proceso determina la manera de desarrollar el producto mientras que la del producto define las propiedades que debe satisfacer el resultante; por otra parte las personas involucradas en los proyectos de software utilizan tecnologías que inciden directamente en la calidad del mismo, debido a que son los llamados a aplicar los procesos para obtener productos de calidad.

En este sentido, el aseguramiento de la calidad del software no radica únicamente en la calidad del producto sino también en la calidad del proceso y de las personas involucradas en el desarrollo; es decir, de las interrelaciones entre las tecnologías, el recurso humano y su estructura.

Para ello, es necesario que se visualice un enfoque sistémico que permita adoptar la calidad orientada a la eficiencia/

efectividad, tanto de los procesos como de los productos y de las personas involucradas, desde las perspectivas del cliente o usuario.

Vale la pena señalar que autores como Spinellis [21] han trabajado el concepto de calidad del software en el contexto de estándares de código abierto. Él afirma que hoy en día existen millones de líneas de código disponible el cual no se le logra aprovechar adecuadamente. Ya se han desarrollado soluciones, a veces ingeniosas y creativas, para una cantidad de problemas pero los programadores prefieren empezar desde cero porque están acostumbrados a escribir código y no a leerlo; con el gran riesgo de estar reinventando la rueda permanentemente.

Al contrario, el trabajo colaborativo de miles de personas en software libre permite desarrollos de calidad mucho más ágiles que el del propietario, por lo que al desplegar un código pequeño y dejarlo disponible para que muchas personas puede aportar mejoras, encontrando errores y añadiendo funcionalidades, permite el crecimiento y mejora del software en menor tiempo y a bajo costo.

V. PROPUESTA

No todos los aspectos de manufactura esbelta pueden ser aplicados al desarrollo de software, en [7] [8] y [14] se ha desarrollado este tema, el objetivo del presente estudio es utilizar esta filosofía para instrumentar herramientas concretas a los personas involucradas en un proyecto para, dentro de este marco general, cumplir con los objetivos de mejorar la calidad sistémica del software, a un tiempo y costo razonable.

5.1. Justificación

Los proyectos de software no cumplen los plazos inicialmente estimados, tampoco se ajustan al presupuesto de recursos asignados, normalmente se obtienen productos de baja calidad, que no cumplen las especificaciones y cuyo código es difícil de mantener. Esto es corroborado en un caso puntual en [11].

Se han propuesto diversos métodos de desarrollo ágil, tal como *Lean Software Development*; el problema es que son filosofías generales, que implican un cambio profundo en la forma de pensar de las personas, en la cultura organizacional, donde el liderazgo juega un rol muy importante; además de la permanente motivación en el personal para lograr uno de sus objetivos primordiales: cambios que permitan mejorar continuamente dando autonomía al personal para tomar decisiones importantes.

De acuerdo a la experiencia de los autores de este artículo tanto en el área industrial, de negocios y académica; impulsar este cambio de paradigma no es nada fácil y requiere de un convencimiento desde los más altos niveles de la organización hasta los más bajos. Dicho cambio se puede alcanzar a través de la sistematización de la forma de hacer las labores del día a día, logrando dar pasos cortos pero continuos hacia la excelencia, impactando positivamente a la organización en forma integral.

Con la propuesta se pretende instrumentar herramientas que sirvan a las personas que trabajen en proyectos que utilicen la filosofía de desarrollo de software esbelto, para coadyuvar a lograr los objetivos finales de obtener productos de calidad, mejorando continuamente a través de la práctica del *kaizén* y el ciclo de calidad de propuesto por Deming: planificar, hacer, verificar y actuar [19].

5.2. Herramientas de manufactura esbelta para el desarrollo de Software de calidad

A continuación se describen cada una de las herramientas y estrategias propuestas que apuntan al logro del desarrollo de software de calidad.

Kanban (tarjeta visual) con un gran pizarrón para monitorear que se está haciendo en cada funcionalidad (diseño, codificación, prueba, compilación y distribución) el equipo de desarrolladores y que está pendiente por hacerse. Para ello se pueden utilizar simplemente notas adhesivas coloreadas (para denotar diferentes tipos de actividades) pegadas sobre el pizarrón, no se necesita usar herramientas tecnológicas de avanzada, aunque tampoco son descartables productos de software de control de proyectos. Lo importante es permitir a cualquier persona perteneciente al equipo de desarrolladores o líderes del proyecto visualizar el estado actual de actividades pendientes en forma integral.

Andon (indicadores de problemas) utilizando un diagrama que muestra el progreso global del proyecto (denominado *Burndown*) e indicando el trabajo restante en el tiempo que denota de un solo vistazo si el proyecto va a tiempo o no. Mantener este tipo de diagramas informativos sobre grandes pizarrones puede proveer información que permite mostrar el estado del desarrollo en un momento dado.

Las cinco S (*Seiri*: organizar, *Seiton*: orden, *Seiso*: limpieza, *Seiketsu*: estandarizar, *Shitsuke*: disciplina) las cuales se explican en detalle a continuación.

- *Seiri:* Organizar las cosas del equipo de desarrolladores en las estaciones de trabajo y servidores. Además, buscar versiones antiguas del software, archivos antiguos y los informes que nunca serán usados más; hacer respaldos de ellos si se debe y luego eliminarlos.
- *Seiton*: son importantes los diseños de escritorio y estructuras de archivos, ellos deben ser elaborados de manera que estén lógicamente ordenados y sean fáciles de encontrar por otra persona. Cualquier lugar de trabajo utilizado por más de una persona debe estar conforme a un diseño común para que se pueda encontrar lo que se necesita cada vez que se tenga que acceder a él. Deben utilizarse estándares para la codificación de programas, nombres (archivos, variables, programas) e interfaces gráficas. Además, establecer políticas de pruebas: definir alcance, frecuencia, forma de automatizarlas y niveles de defecto.

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

- *Seiso*: No debe haber restos de comida, tazas de café, vasos; se debe limpiar las huellas dactilares en las pantallas y recoger todo papel de desperdicio, también la pizarra debe estar limpia después de tomar de allí los diseños importantes que fueron esbozados.
- *Seiketsu*: Colocar las herramientas de automatización y estándares en su lugar para asegurarse de que las estaciones de trabajo tienen siempre la última versión, efectuar copias de seguridad periódicamente y eliminar archivos basura misceláneos que no deben ser acumulados.
- *Shitsuke*: En este punto sólo hay que mantener la disciplina establecida. Lo importante para proyectos de desarrollo de software es el establecimiento del orden, sistematización, limpieza y estándares para mantener luego esta disciplina.

Heijunka (nivelar cargas de producción), nivelar el trabajo de las personas estableciendo un balance entre las labores diarias y las inherentes al proyecto. Esto se puede lograr mediante una adecuada planificación y seguimiento a través de herramientas automatizadas de control de proyecto.

Poka Yoke (a prueba de errores), a través de la especificación del código mediante aserciones en los programas (sirven además como documentación), permite a los programadores implementar la verificación de los errores antes de que ocurran en lugar de hacer énfasis en la inspección (control posterior). Por otra parte se deben utilizar en lo posible las restricciones en los sistemas gestores de base de datos para forzar la validación y limpieza de los datos ingresados por los usuarios.

Value Stream Map (Mapa del flujo de Valor), una de las formas de eliminar las esperas (desperdicio) es analizar el mapa de la cadena de valor del proceso de desarrollo de software, el cual puede hacerse siguiendo los pasos que se detallan a continuación:

1. Seguir el proceso de desarrollo de software desde la perspectiva de un cliente (tantos externos como internos), buscando los datos de acuerdo a lo vivido y observado.
2. Anotar el tiempo promedio que el equipo invierte en cada actividad.
3. Abajo del mapa, dibujar una línea de tiempo en la que se separe el tiempo de Valor y el tiempo de Espera.
4. Revisar las actividades con mayor tiempo de espera e identificar las fuentes del retraso o esperas y definir cómo reducir dicho tiempo.

Doxygen (http://www.doxygen.org) herramientas generadoras de documentación para código fuente para que sea la mínima requerida. Una excesiva documentación consume recursos, produce demoras, se pierde y se convierte en obsoleta muy rápidamente.

Feature team, formar equipos de trabajo orientados a las funcionalidades de acuerdo a lo propuesto por Larman [8], asignando responsabilidades de acuerdo a las habilidades de las personas. Dicho equipo tiene que diseñar, codificar y probar la funcionalidad, además de entregar un producto de calidad que cumpla con los requerimientos del cliente según la característica desplegada. Las personas van rotándose en sus roles, por lo que van aprendiendo a hacer todas las actividades que involucran el desarrollo del proyecto.

Causa raíz del problema, usar técnicas como la de preguntarse cinco veces ¿por qué? al detectarse un error, el cual debe ser corregido, luego de ser sometido a prueba el software y actualizada la versión en producción. Los errores de software (*bugs*) son una gran fuente de desperdicio, cuyo porcentaje se puede medir como el impacto del mismo por el tiempo sin ser detectado.

Programación por pares y revisiones de diseño, centrándose en amplificar el aprendizaje y compartir las experiencias, más que en exacerbar los errores de cada persona.

Prototipos, presentando a los usuarios los avances del software antes de que todos los detalles del diseño hayan sido finalizados, obteniendo de esta forma una retroalimentación continua y evitando desarrollar características extras.

Estándares de codificación para una buena gestión del código fuente, esto permite que el software sea autodocumentado.

Automatizar disciplinas básicas de programación, tales como: establecer controlador de versiones (mantener todo el código fuente y otros artefactos del proyecto en un repositorio central con un historial completo de versiones de cada archivo), efectuar integraciones constantes de cada funcionalidad entregada y realizar pruebas periódicas. Para ello se pueden utilizar herramientas de software libre disponibles en la web tales como:

- CVS (http://www.cvshome.org) o SVN (http://subversion.tigris.org)
- ANT (http://ant.apache.org),
- X-Unit (http://xprogramming.com/software)
- FIT (http://fit.c2.com ó http://www.fitnesse.org)
- Validadores para aplicaciones web (http://www.w3.org/Status.html).

Otras herramientas que pueden utilizarse derivadas de los principios del desarrollo esbelto son: eliminar desperdicios (códigos o funcionalidades innecesarias, retrasos en el desarrollo, requerimientos imprecisos, burocracia, comunicación interna lenta), amplificar el aprendizaje (entrenar continuamente a desarrolladores y líderes en aspectos como el trabajo en equipo, liderazgo, toma de decisiones, empoderamiento), decidir lo más tarde posible siempre que sea un momento responsable (para hacerlo con la mayor información disponible), entregar avances lo más rápido que sea posible, facultar al equipo de trabajo, integración continua de los avances y utilizar un

enfoque sistémico (tener una visión del todo y como las partes influyen en él).

En la Figura 2 se muestra la metodología resultante de la aplicación directa del ciclo de mejora de calidad de Deming, mediante un ciclo iterativo e incremental de cuatro pasos: planificar, hacer, verificar y actuar.

Figura 2. Metodología de Desarrollo de Software Propuesta

La planificación se efectúa a través de un proceso de desarrollo centrado en las personas; incorporando las herramientas de manufactura esbelta al desarrollo de software propuestas (hacer); luego el control de los diversos productos, procesos y personas que intervienen en el proyecto (verificar) mediante modelos como MOSCA [12]; en base a la evaluación obtenida, los líderes del proyecto tienen que aplicar correctivos (actuar) para el mejoramiento continuo (*kaizén*).

VI. CONCLUSIONES Y RECOMENDACIONES

La filosofía de pensamiento esbelto se basa en el sentido común y propone metodologías sencillas de mejora, participación de los trabajadores en ella, rendimiento y calidad en los recursos productivos en los que se aplica, lo que permite detectar e identificar muchas fuentes de perfeccionamiento en el trabajo mediante la eliminación o reducción del desperdicio.

Los proyectos de software deben seguir los principios aquí expuestos para obtener productos lo más cercano a lo requerido por los usuarios, incorporando procesos de calidad y personal que esté motivado a mejorar continuamente la forma de hacer el trabajo para que incida en la calidad sistémica de desarrollo.

En particular, además de la filosofía global, tal como el desarrollo de software esbelto (*Lean Software Development*) que debe ser permanentemente seguida por los líderes del proyecto y compartida por las personas involucradas, se deben utilizar puntualmente herramientas de manufactura esbelta tales como: *Kanban*, *Andon* las 5'S, *Heijunka, Poka Yoke* y *Value Stream Map*.

Emplear otras herramientas propias de metodologías ágiles, que permitan eliminar fuentes de desperdicio: *feature teams*, documentación mínima requerida, la causa raíz de los errores (*bugs*), programación por pares, revisiones de diseño, implementación de prototipos, establecer estándares de codificación; finalmente, automatizar disciplinas básicas de programación.

El equipo del proyecto se debe enfocar en cumplir con los plazos, la cantidad de características añadidas al software, poder realizar cambios en los requerimientos del cliente (flexibilidad en el desarrollo), lo que producirá una reducción en el ciclo de vida, sin concentrarse en optimizar los subsistemas de medición e inspección posterior. En cuanto a los líderes del proyecto, estos deben ser permanentes motivadores de los desarrolladores, empoderar a los mismos y deben promover la amplificación del conocimiento.

Como trabajos futuros se destaca llevar a la práctica esta propuesta en proyectos de desarrollo de software en un entorno científico-académico, evaluando la mejora en la calidad sistémica de desarrollo del software. También puede compararse la efectividad de esta propuesta con respecto a la aplicación de otros métodos ágiles (*XP*, *Scrum* y método *Crystal*).

BIBLIOGRAFÍA

[1] Castaño, C., Maíz, I., Palacio, G. y Villarroel, J., 2008. Prácticas Educativas en Entornos Web 2.0. Madrid, España: Editorial Síntesis. 196 P.

[2] Dingsøyr, T., Dybå, T. y Abrahamsson, P., 2008. A Preliminary Roadmap for Empirical Research on Agile Software Development. En: Agile 2008 Conference. DOI 10.1109/Agile.2008.50, pp. 83-94.

[3] Dybå, T. y Dingsøyr, T., 2008. Empirical studies of agile software development: A systematic review. En: Information and Software Technology, 50, 883-859.

[4] Feld, W., 2000. Lean Manufacturing: tools, techniques, and how to use them. The St. Lucie Press/APICS Series on Resource Management. 228 P.

[5] Fogelström, N.D., Gorschek, T., Svahnberg, M. y Olsson, P. 2010. The Impact of Agile Principles on Market-Driven Software Product Development. En: Journal of Software Maintenance and Evolution: Research and Practice. Wiley InterScience. DOI: 10.1002/spip.420. 22 (1), pp. 53-80

[6] Fritzsche, M. y Keil, P., 2007. Agile Methods and CMMI: Compatibility or Conflict? En: e-Informatica Software Engineering Journal, 1(1), pp. 9-26.

[7] Hibbs, C., Jewett, S. y Sullivan, M., 2009. The Art of Lean Software Development. USA: O'Reilly Media Inc. 128 P.

[8] Larman, C. y Vodde, B. 2008. Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum. EEUU: Addison Wesley. 348 P.

[9] Lei, H., Claus, M., Rammage, R., Baer, C.D., Decool, R., Kniss, J., Clyde, S., Cooley, D. y Liu, D., 2009. Software's Eight Essentials.

En: International Conference on New Trends in Information and Service Science (NISS '09), Beijing, China, pp 1203 - 1208.
[10] Ohno, T., 1988. Toyota Production System. Beyond Large-Scale Production. New York: Productivity. Inc. 143 P.
[11] Omaña, M. y Cadenas, J., 2010. Manufactura Esbelta: una contribución para el desarrollo de software con calidad. En: Enl@ce Revista Venezolana de Información, Tecnología y Conocimiento, 7 (3), pp. 11-26.
[12] Pérez, M.; Domínguez, K.; Mendoza, L. y Grimán, A., 2006. Human Perspective in System Development Quality. En: Proceedings of the Twelfth American Conference on Information Systems American Conference on Information Systems - AMCIS2006. Acapulco, México, pp. 3.823-3.834
[13] Petersen, K. y Wohlin, C., 2010. Software process improvement through the Lean Measurement (SPI-LEAM) method. En: The Journal of Systems and Software, 83, pp. 1275-1287.
[14] Poppendieck, M. y Poppendieck, T., 2009. Leading Lean Software Development: Results are not the Point. Addison Wesley. 278 P.
[15] Poppendieck, M., 2007. Lean Software Development. En (tutorial): 29th International Conference on Software Engineering (ICSE'07 Companion), Minneapolis, EEUU, pp. 165-166.
[16] Poppendieck, M. y Poppendieck, T., 2007. Implementing Lean Software Development: From Concept to Cash. EEUU: The Addison-Wesley Signature Series. 276 P.
[17] Poppendieck, M. y Poppendieck, T., 2003. Lean Software Development: An Agile Toolkit. EEUU: Addison-Wesley. 203 P.
[18] Pressman, R., 2005. Ingeniería del Software - un enfoque práctico. España: Mc Graw-Hill, Sexta Edición. 958 P.
[19] Scholtes, P., Joiner, B. y Streibel, B., 2003. The Team Handbook. EEUU: Joiner/Oriel. Third Edition. 400 p.
[20] Sommerville, I., 2006. Ingeniería del Software. Madrid: Pearson Educación, Septima Edición. 687 P.
[21] Spinellis, D., 2006. Code Quality: The Open Source Perspective. Effective Software Development Series. Boston, EEUU: Pearson Education. 569 P.
[22] Staron, M., Meding, W. y Söderqvist, B., 2010. A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation. En: Information and Software Technology, 52, pp. 1069-1079.
[23] Stepanek, G., 2005. Software Project Secrets: Why Software Projects Fail. EEUU: Apress. 166 P.
[24] Tian, J., 2005. Software Quality Engineering. Testing, Quality Assurance and Quantifiable Improvement. EEUU: IEEE Computer Society Press, Wiley-Interscience. 440 P.
[25] Udo, M, Vaquero, T. S., Silva, J.R. y Tonidandel, F., 2008. Lean Software Development Domain. En: Proceedings of ICAPS 2008 Scheduling and Planning Application workshop. Sydney, Australia.
[26] Voas, J., 1999. Software Quality's Eight Greatest Myths. En: Software, IEEE. 169 (5), pp. 118-120.
[27] Womack, J. y Jones, D., 2005. Lean Thinking: Cómo utilizar el pensamiento Lean para eliminar los despilfarros y crear valor en la empresa. España: Ediciones Gestión 2000. 478 P.
[28] Womack, J., Jones, D. y Roos, D., 1991. The Machine that Changed the World: The Story of Lean Production. EEUU: Harper Perennial. 323 P.