Lean manufacturing aplicado a la Ingeniería de *Software*

Herman Camacho Sánchez, Anthony Fuentes Fallas, Prof. Miguel Corea

Escuela de Ingeniería,
Universidad Latinoamericana de Ciencia y Tecnología,
ULACIT, Urbanización Tournón, 10235-1000
San José, Costa Rica
[hcamachos289,afuentesf897]@ulacit.ac.cr
http://www.ulacit.ac.cr

Resumen La Ingeniería de *Software*, en ocasiones, genera prácticas y procesos que no agregan valor a los productos y servicios que se le entregan a los usuarios, los cuales incurren en desperdicios, como por ejemplo el retrabajo, procesos redundantes, productos desechados por no tener la calidad necesaria, entre otros. También sucede que sus procesos no son lo suficientemente esbeltos como para producir más con la menor cantidad de recursos. Es así como se incrementan los costos, se disminuye la satisfacción de los usuarios, así como la calidad y la imagen de TI. En el presente artículo se presenta un análisis de cómo *lean manufacturing* puede ser una herramienta útil en el Desarrollo de *Software*, obteniendo beneficios al optimizar los procesos productivos, aplicando principios claves como la reducción del desperdicio, la optimización de las corrientes de valor, fomentar la mejora continua, motivar al personal involucrado e incrementar la productividad.

Keywords: *Lean*, Desarrollo de *Software*, Optimización, Reducción de Costos, Productividad

1. Introducción

A pesar de que el término *lean*, como tal, proviene tanto del proceso de gestión de producción como del desarrollo de productos en el MIT, el término *lean manufacturing* nació en la empresa fabricante de autos Toyota. Este fue el resultado de la búsqueda de la optimización de sus sistemas de producción al reducir los inventarios, hacer que los operarios trabajaran con más máquinas y que a la vez se le diera más importancia a la calidad, opuesto a los sistemas de producción masivos basados en técnicas de *"buffers"* del productor americano de autos Ford, la cual establece un nivel aceptable de defectos y niveles máximos de inventarios. (Poppendieck, 2007)

Mientras tanto, la Ingeniería de *Software*, a través del tiempo, ha tratado de hacer uso de diversas técnicas y herramientas para hacer sus procesos más flexibles, eficientes y que brinden un mayor valor. Un ejemplo de estas iniciativas ha sido el uso de *Metodologías Ágiles*.

Sin embargo, si el Desarrollo de *Software* no hace uso de técnicas, herramientas y metodologías adecuadas que le agreguen valor tanto a los clientes internos como externos, sus procesos serán poco eficientes y consumirán más recursos entregando menos resultados. Esto genera, como consecuencia, un incremento en los costos, insatisfacción de los usuarios y clientes, reducción de la calidad, demoras en las fechas de entrega de los productos, falta de documentación de las pruebas en el desarrollo, una mala gestión de cambios en los requerimientos y la creación de funcionalidades que no le agregan valor a los productos de *Software*.

Por lo anterior, el presente artículo hace un análisis para explicar los principales problemas a la hora de realizar el Desarrollo de *Software*, a la vez que se propone la incorporación de *Lean Manufacturing* como herramienta de mejora tanto en ambientes que usen Metodologías Ágiles como en ambientes que no las usen (metodologías tradicionales), basándose en los aspectos esenciales estipulados en *lean manufacturing* que pueden aplicarse tanto en los procesos a nivel técnico como de gestión de la Ingeniería de *Software*.

Para cumplir con el fin de la investigación, esta incorporará los problemas más comunes en el Desarrollo de *Software* que reducen la eficiencia, se enfocará de manera general en los principios de *lean manufacturing* que son aplicables al Desarrollo de *Software*, tomando en consideración las metodologías ágiles *Scrum* y XP utilizadas en la industria actual, así como escenarios que no hacen uso de metodologías ágiles, para relacionarlas con la implementación de los principios de *lean manufacturing*.

Como objetivo principal de la investigación, se pretende analizar la incorporación de *lean manufacturing* en el desarrollo de *Software* para reducir desperdicios, aumentar la eficiencia, la productividad y la satisfacción de los clientes con productos de calidad, asociando metodologías ágiles y convencionales.

Los objetivos secundarios presentes en la investigación son: determinar cuáles son los problemas más comunes que disminuyen la eficiencia en el Desarrollo de *Software*, explicar por qué los principios de *lean* se consideran efectivos tanto a nivel técnico como a nivel de gestión y que pueden ayudar a optimizar procesos y actividades en la Ingeniería de *Software*, investigar cómo se podría complementar el uso *lean manufacturing* en ambientes de Desarrollo de *Software* que usen Metodologías Ágiles como *Scrum* y XP, así como en ambientes que usen metodologías convencionales.

Dentro de este contexto, la investigación se enfoca en atacar la pregunta clave de ¿cómo se aplica *lean manufacturing* al Desarrollo de *Software* de manera que permita optimizar actividades y procesos al reducir desperdicios, aumentar la eficiencia, el valor, la calidad, la productividad y la satisfacción general de los clientes?

Esto, a la vez, produce otras preguntas secundarias requeridas para contestar la pregunta principal, como son: ¿Por qué se requiere analizar la aplicación de *lean manufacturing* al Desarrollo de *Software*? ¿En qué consiste la aplicación de *lean manufacturing* en el Desarrollo de *Software*? ¿Cómo se complementará *lean manufacturing* con otras Metodologías Ágiles, tales como *Scrum* y XP, en el Desarrollo de *Software*?

Para responder lo anterior, el método utilizado para la obtención de resultados y la elaboración de las recomendaciones se basó en el estudio y revisión bibliográfica de los conceptos, principios, pilares e ideales de *lean manufacturing* mediante el uso del buscador *EBSCO Discovery Service*, además del análisis de las metodologías ágiles *Scrum* y XP, la metodología en cascada y algunos de los problemas más comunes en los procesos de Desarrollo de *Software*, mediante casos de estudio. Cada artículo fue revisado por resumen e introducción, para determinar si el contenido es relevante con respecto a la relación existente entre los principios de *lean manufacturing* con la Ingeniería de *Software*.

Además, cabe resaltar que a la fecha se han realizado varias publicaciones relacionadas con la aplicación de *lean manufacturing* en el Desarrollo de *Software*, la mayoría basadas en el libro *"Lean Software Development: An Agile Toolkit"* publicado por Mary Poppendieck y Tom Poppendieck (Poppendieck y Poppendieck, 2003). Otros libros publicados por los autores son *"Lean Software Development: An Agile Toolkit for Software Development Managers"* y *"Implementing Lean Software Development: From Concept to Cash"*.

2. Marco Teórico

Luego de la Segunda Guerra Mundial, la demanda en la economía en Japón fue baja, por lo que la necesidad de producir más con menos recursos para mantener los costos bajos era una necesidad para la empresa productora de autos Toyota. Esta empresa ha sido una gran promotora de filosofías para la mejora continua de los procesos conocidos en Toyota como *kaizen*, el aumento de la calidad y la producción eficiente de vehículos basados en varios principios, entre ellos los del pensamiento *lean* (Roel, 1998; Liker, 2010).

A pesar de ser una metodología que fue concebida inicialmente para procesos de manufactura, se ha adaptado al Desarrollo de *Software*. Algunos conceptos importantes que se recomiendan conocer antes de evaluar los resultados y conclusiones presentados son:

2.1. Tecnologías de Información

Este concepto surgió a mediados del siglo XX, después de la Segunda Guerra Mundial, desarrollando notaciones lógicas para el procesamiento de información mediante circuitos eléctricos, la máquina de Turing, ENIAC y la capacidad para enviar tramas de datos a través de redes de computadoras.

La literatura lo define como el uso de computadoras, equipos de telecomunicaciones, *hardware*, *software* y dispositivos periféricos para el procesamiento, protección, almacenamiento, recolección y recuperación de los datos. En la actualidad, esto es posible mediante el uso de un abanico muy amplio de soluciones, productos y servicios disponibles en el mercado.

Como Departamento de Soporte al Negocio, una de las expectativas hacia este es la capacidad de gestionar cada vez mayor información, de manera más flexible, con nuevas funcionalidades y características requeridas por los usuarios finales

o clientes para lograr los objetivos de las organizaciones. Debido a esto, *lean manufacturing* puede implementarse de una forma muy similar a como se emplea en las plantas de manufactura, ya que aspectos claves como la reducción del desperdicio, la operación de varios procesos por parte de un operador (similar a operar varias máquinas en una planta de manufactura), la información adecuada entre el flujo de procesos, entre otros, se pueden usar dentro del plan de mejora de las actividades de T.I.

2.2. Gestión de la Calidad

La Gestión de la Calidad es todo un conjunto de normas, procedimientos, estándares y políticas que hacen uso de diversas herramientas, con el fin de lograr que una organización pueda evaluar y mejorar la calidad de sus procesos y su Sistema de Gestión de Calidad, a la vez que se hace énfasis en la búsqueda continua de la calidad.

Los objetivos principales de una efectiva Gestión de la Calidad son lograr y sobrepasar la satisfacción de sus *clientes*, los cuales son las personas que harán uso de los productos o servicios brindados por la organización. Estos pueden ser internos o externos. Los *clientes internos* son los que pertenecen a la misma organización, mientras que los *clientes externos* son los que harán uso de los productos pero que no pertenecen a la organización.

Para lograrlo, las organizaciones se dedican a gestionar la calidad para lograr ventajas competitivas que puedan mantener o superar una competencia cada vez más exigente en el mercado, en el cual se demandan mejores productos y servicios libres de defectos.

En el mercado existe una gran variedad de estándares de gestión de la calidad que han sido globalmente aceptados, los cuales han sido creados y definidos por diferentes organizaciones normalizadoras, tal es el caso de ISO, DIN, EN, entre otras. Uno de objetivos de estos estándares es que las organizaciones puedan utilizarlos como instrumento de auditoría, por parte de alguna entidad autorizada, para validar y verificar si los procesos que se están realizando internamente cumplen con las especificaciones indicadas en su sistema de gestión de la calidad.

Lean manufacturing se basa en prácticas sólidas de Gestión de Calidad de diversas formas, siendo una de las más importantes el hecho que cada operador se asegure que los productos o servicios que resulten de su proceso, sigan su flujo sin defectos. Esto se logra al asegurarse de que el objetivo sea generar productos o servicios de calidad, reparando sus propios defectos, y no hacer una gran cantidad de productos que conlleven una tasa de defectos que puedan afectar los procesos siguientes (o estaciones siguientes en las plantas de manufactura).

2.3. Ingeniería de *Software*

La *Ingeniería del Software* trata de la aplicación del conocimiento científico, de principios y metodologías para lograr enfoques sistemáticos, disciplinados y cuantificables, por medio de técnicas, herramientas y métodos para el desarrollo, operación y mantenimiento de *Software*. Adicionalmente se incluye el análisis

y diseño del proyecto o producto, el desarrollo del *software*, las pruebas para confirmar que cumple con los criterios de aceptación de quienes solicitaron los requerimientos y finalmente, la implementación del producto de *software*. (Leach, 1999).

Como parte de la búsqueda de técnicas y métodos, el término de agilidad empezó a surgir desde los años noventa; sin embargo, se adoptaron hasta el año 2001 mediante el manifiesto ágil, del cual surgen metodologías tales como *Scrum*, *Extreme Programming (XP)* y *lean*. Por lo tanto, contribuyen a comprender mejor el producto que se pretende realizar, a generar mejores decisiones en el proceso de desarrollo de un *software*, siendo esto posible con la participación total de un equipo de trabajo.

Los cuatro valores principales de las metodologías ágiles son:

- Individuos e interaciones sobre procesos y herramientas.
- *Software* funcional sobre documentación excesiva.
- Colaboración del cliente sobre negociaciones al contrato.
- Respuesta al cambio sobre seguimiento de un plan.

También define un conjunto de interesados con un responsable dedicado a colaborar activamente en la aclaración de los requerimientos, establecer prioridades sobre las actividades que debe realizar el equipo, planificar los entregables según los tiempos y costos en iteraciones cortas y asegurar la calidad en los procesos, mediante criterios de aceptación y la fase de retrospectiva, ayudando a mejorar continuamente el desempeño del equipo de trabajo a lo largo de la ejecución del proyecto (Vallon, Wenzel, Brüggemann, y Grechenig, 2015; Schön, Escalona, y Thomaschewski, 2015).

Otros métodos aplicados a la Ingeniería de *Software* son los convencionales, los cuales han sido empleados empíricamente en las organizaciones. Bajo este esquema existe incertidumbre sobre el producto que se pretende entregar, carece de controles rigurosos que den seguimiento a las actividades de un proyecto y el personal interesado carece de una planificación para solicitar nuevos requerimientos o mejoras, afectando negativamente la mantenibilidad del entregable, la productividad y la calidad, incrementando costos y tiempo.

2.4. *Lean Manufacturing*

Lean Manufacturing nace en las plantas de manufactura de Toyota como parte de las iniciativas de optimización de las líneas de producción de autos. El concepto de *Lean Manufacturing* era un enfoque opuesto al que empleaban los tres productores más grandes de vehículos en Norteamérica, que se basaban en técnicas de *"buffering"*, las cuales consistían en producir en exceso para compensar ciertos niveles de defectos. En contraste, el enfoque de *Lean Manufacturing* estaba basado en la mayor producción con menos recursos por medio de procesos optimizados con un flujo constante y sin desperdicios.

Los siete Principios de *Lean Manufacturing* son:

1. Mejora continua en la calidad.

2. Apoderar trabajadores.
3. Trabajadores capacitados.
4. Prevención de defectos.
5. Mediciones simples y visuales de la calidad.
6. Dispositivos de medición automática de la calidad.

Lean manufacturing va en contra del exceso de inventario, ya que este factor oculta problemas tales como una planificación pobre del tiempo, fallos en las máquinas, transporte, entre otros. Todos los empleados son partícipes de la metodología y poseen un conocimiento general y las habilidades para eliminar los desperdicios.

También se realiza un proceso de aseguramiento de la calidad en el que los procesos se detienen en caso de surgir alguna anormalidad, tales como problemas en los equipos, problemas de calidad, material retrasado, etc.

Como complemento a *Lean manufacturing*, *Just in Time* se refiere a la práctica de producir y transmitir solamente lo que se necesita y cuándo se necesita en la cantidad requerida.

Uno de los principios establecidos en *lean* menciona el concepto de *desperdicio*, el cual se relaciona con las actividades o tareas que no le agregan valor a los procesos y que a su vez afectan los productos y servicios que la empresa brinda. Básicamente un desperdicio es lo contrario a un proceso que genere valor al cliente.

Dentro de los principales tipos de *desperdicio* clasificados a nivel de manufactura están:

1. La sobreproducción relacionada principalmente cuando se produce más de la cuenta o demasiado pronto.
2. El transporte que no es esencial.
3. Mayor inventario que el mínimo para realizar el trabajo.
4. Los tiempos de espera por alguna pieza para poder completar el ciclo de la máquina.
5. El sobre procesamiento, sobre trabajo o reparaciones.
6. Cualquier movimiento innecesario en el flujo que no agregue valor.
7. La producción de defectos que consuma el tiempo y talento de las personas.

En el *Desarrollo de Software* hay malas prácticas y errores muy comunes en los procesos que perjudican la productividad y que reducen valor en los productos, mientras que aumentan otros aspectos, tales como funcionalidades no requeridas por los usuarios, requerimientos no cumplidos, retrabajos, entregables con bajos niveles de calidad, entre otros. Uno de los fundamentos más importantes de *Lean Manufacturing* es el de detectar y eliminar, en la medida de lo posible, todos los desperdicios en todos los procesos productivos.

Mary y Tom Poppendieck (Poppendieck y Poppendieck, 2003) modificaron los principios originales de *Lean Manufacturing* y crearon los siguientes siete Principios de *Lean Software Development*:

1. Eliminar desperdicios.

2. Crear conocimiento.
3. Decidir lo más tarde posible.
4. Entregar lo más pronto posible.
5. Respetar a las personas.
6. Integrar la calidad.
7. Optimizar el "todo".

Finalmente establecieron los 7 tipos de *desperdicios* orientados al desarrollo de *software*, los cuales son:

1. Trabajo parcialmente completado (equivalente a "inventario en proceso").
2. Funcionalidades extra (equivalente a "sobre producción").
3. Los "traspasos" de conocimiento (equivalente a l "transporte no esencial").
4. Intercambio de tareas (equivalente a "movimientos innecesarios").
5. Los retrasos (equivalente a "tiempos de espera").
6. El reaprendizaje o el retrabajo (equivalente al "sobreprocesamiento").
7. Los defectos (el mismo en el área de manufactura).

3. Resultados

Con base en la revisión sistemática de los artículos investigados y los casos de estudio, además de un análisis propio basado en la experiencia en el área de Desarrollo de *Software*, los problemas más comunes que se encontraron fueron:

3.1. Reuniones no efectivas

- Se da una pobre definición de la agenda.
- Tiempo gastado en actividades ajenas al proyecto.
- Se invierte mucho tiempo en un solo tema,.
- Se reabren problemas cerrados,.
- Falta de decisiones.
- Falta de enfoque en el cliente.
- Falta de predicción de obstáculos.

3.2. Pobre comunicación

- Se acepta todo lo que el cliente quiera sin abrir un diálogo constructivo con un *feedback* significativo.
- Falta de motivación en el equipo.
- Tomar asuntos de manera personal.
- Pobre comunicación entre clientes, desarrolladores y usuarios.

3.3. Problemas de integración

- Producen fallos y caídas, defectos o errores en las aplicaciones.
- Son afectados por factores ambientales.
- Fallos en la infraestructura o en el *software*, virus, *hacker*, *hardware*, red o errores en el operador.

3.4. Falta de conocimiento

- Cuando los desarrolladores carecen de las capacidades necesarias.
- Falta de ayuda, por parte de los desarrolladores más experimentados, ocasionando desperdicios de tiempo.

3.5. Calidad del código

- Se crea código basado en cantidad sobre calidad.
- Se coloca más personas para resolver tareas complejas, no soluciona el problema y retrasa el proceso, aumenta los costos, reduce la calidad, aumenta la posibilidad de problemas de comunicación y dificulta la integración del código.
- Se dan pobres métricas de evaluación.
- El código excesivamente dinámico, con grandes cantidades de código que sea difícil de mantener.
- Pruebas pobres y pocas mediciones relevantes.
- No cumplen con los requerimientos basándose en costos, tiempo, calidad y objetivos planificados.

3.6. Pobre organización

- Se da una pobre priorización entre defectos y nuevas funcionalidades.
- Mala planificación del tiempo, en donde se pierde el *"deadline"*, se incluye código sin probar y, por ende, se reduce la calidad.
- Pobre liderazgo.
- Pobre estimación.
- Falta de recursos o recursos inapropiados.
- Falta de participación de los usuarios.
- Cambios en los requerimientos sin planificación ni control.
- Metas no realistas.
- Pobres estimaciones de recursos necesitados.
- Pobre gestión de proyectos.
- "Politicas" de los interesados.
- Falta de participación de los interesados.
- Presiones económicas.
- Pobres informes del estado del proyecto.
- Riesgos no gestionados.

3.7. Pobre documentación

- Como un ejemplo, cuando los desarrolladores deben arreglar un defecto o implementar una nueva funcionalidad sobre un código que ellos no crearon, se produce un desperdicio al revisar qué es lo que el código realiza, en vez de repararlo más rápidamente.

- Se crean retrabajos al desarrollar código redundante por desconocimiento de lo existente.
- Requerimientos de sistema pobremente definidos, incompletos.

Comparando los problemas mencionados anteriormente, se realizó un análisis para determinar las causas, con el fin de establecer recomendaciones basadas en los tipos de desperdicios que señala *Lean Software Development*, además de asociarlas con las *metodologías ágiles* y las *metodologías convencionales*:

1. Trabajo parcialmente completado (equivalente a "inventario en proceso"): en este tipo de desperdicio se da los problemas más graves, ya que no se tiene un código aprobado y sin errores, o *"bugs"*.
   Causas:
   - Código completado pero pendiente de aprobar por medio de los "testers".
   - Código sin documentar.
   - Código aún no liberado a producción.
   - Exceso de comentarios en el código.
   - Demasiado "trabajo en progreso" (*Work in Progress*) hace que las nuevas funcionalidades por desarrollar no puedan realizarse aún.

   Recomendaciones:
   - El código debe estar listo para que los encargados de usarlo en la siguiente fase puedan trabajar. Se recomienda tener el código listo de una manera *"Just in Time"*.
   - Deben eliminarse los çuellos de botella"para aumentar el flujo entre los procesos.
   - Reducir el código no probado y así reducir la posibilidad de tener ïnventario de código"que tenga defectos y que pueda afectar los tiempos de entrega o la calidad.
   - Reducción de la duración de los ciclos.

   Metodologías Ágiles:
   - En el caso de *scrum* se deben tener claros los requerimientos y el valor por medio de historias de usuario.
   - Debe reforzarse la claridad del *Backlog*, de manera que puedan apreciarse las dependencias entre las historias de usuario.

   Metodologías Convencionales:
   - Debe tenerse atención especial a los requerimientos de los usuarios y no enfocarse solamente en recolectarlos al inicio del análisis.
   - Deben definirse correctamente los entregables, los criterios de aceptación y quiénes serán los involucrados en tales procesos.
2. Funcionalidades extra (equivalente a "sobreproducción"): en este tipo de desperdicio, un porcentaje importante de las funcionalidades no serán usadas por los usuarios, agregando complejidad al código.
   Causas:
   - Se desperdicia tiempo y recursos en funcionalidades que no agregan valor al producto de *software*.
   - Se desconoce o existe incertidumbre en conocer los clientes meta.

- Crear código nuevo cuando ya existe código disponible y probado, tal como *caches*, bases de datos, *pool de threads*, *pool de conexiones*, gestor de transacciones.

Recomendaciones:

- Si se quiere entregar un valor agregado, debe tenerse en cuenta que una buena comunicación con el cliente e interesados, hacen que estas funcionalidades sean utilizables en el futuro.

Metodologías Ágiles:

- En *scrum*, debe confirmarse el valor de las entradas en el *Backlog* para determinar que sean de valor para el cliente.
- Deben usarse los indicadores de valor, costo y riesgo para priorizar las funcionalidades del producto.

Metodologías Convencionales:

- Debe producirse lo requerido, en el momento en que es necesitado.

3. Los "traspasos" (equivalente al "transporte no esencial"): es una de las causas por las cuales se pierde tiempo y recursos tratando de comprender lo que otro desarrollador realizó.

   Causas:

   - Traspasos innecesarios de código entre desarrolladores.
   - El conocimiento original se va perdiendo en una mayor frecuencia.
   - Traspasos innecesarios de código entre el desarrollador y el *"tester"*.
   - Traspasos innecesarios de código de la etapa de desarrollo a la de *"deployment"*.

   Recomendaciones:

   - El equipo de trabajo debe trabajar todo el tiempo juntos, para que los desarrolladores estén familiarizados con el código de los demás.

   Metodologías Ágiles:

   - Se debe reforzar la idea de tener equipos multifuncionales para que se reduzcan los traspasos.
   - Si los equipos no están juntos físicamente se debe al menos hacer el reporte del *"handoff"*.
   - Promover que el equipo sea autosuficiente y exista una comunicación transparente.

   Metodologías Convencionales:

   - Generar reuniones para realizar revisiones de código, de forma periódica, con el fin de asegurar que los desarrolladores estén familiarizados con el código existente.

4. El reaprendizaje o el retrabajo (equivalente al "sobreprocesamiento"): son los atrasos y el aumento en los costos al invertir tiempo reparando problemas cuando el código no llevaba los niveles de calidad necesarios.

   Causas:

   - Pobre planificación.
   - Pobres requerimientos que conllevan a un diseño y desarrollo pobre de código.
   - Pobre calidad.
   - Intercambio de tareas.

- Gobernanza innecesaria.
- Pruebas inapropiadas.
- Se requiere simplificar y optimizar los procesos.
- Pobre comunicación.
- Código no documentado.
- Tener actividades o procesos que no agregan valor, aumentando el tiempo, reduciendo el progreso y la productividad.

Recomendaciones:

- Planificar de manera que las actividades se realicen correctamente, o al menos que se reparen, antes de pasar el código a la siguiente persona.
- Trabajar en equipo y compartir el conocimiento entre los miembros de los equipos para asegurar que el conocimiento esté balanceado.
- Contar con un repositorio centralizado de documentos para que la información esté al alcance de todos, en cualquier momento.

Metodologías Ágiles:

- A nivel de *scrum*, la documentación debe estar anexada a las historias de usuario.
- Invertir tiempo en compartir conocimientos útiles en cada uno de los *sprints*, mediante la retrospectiva o en reuniones de carácter formal, tales como los cierres o inicios de iteración.
- Si las personas no están en la misma ubicación, deben implementarse herramientas que reduzcan el problema de la no presencia física, tales como videoconferencias, etc.

Metodologías Convencionales:

- Cuando se realiza el análisis antes del diseño y se requiere realizar cambios, se produce demasiado retrabajo, por lo que debe implementarse una gestión de cambios.
- Tener una adecuada comunicación con los interesados para estar todos "sincronizados".

5. Defectos: es uno de los desperdicios que más afecta los proyectos de desarrollo de *software*, por lo tanto, deben mantenerse al nivel más bajo posible o reducirlos a cero.

   Causas:

   - Falta de comprensión de los requerimientos.
   - Falta de pruebas efectivas sobre el código.
   - Falta de criterios de aceptación.
   - Falta de conocimientos técnicos en los miembros del equipo.
   - Los *"testers"* no están involucrados en los procesos.
   - Falta de documentación de casos de uso.

   Recomendaciones:

   - Realizar pruebas manuales y automatizadas, con base en los criterios de aceptación.
   - Integración de código aprobado y liberación a producción lo más rápido posible.
   - Involucrar a los recursos dedicados a pruebas.
   - Se debe compartir los conocimientos entre los miembros del equipo.

- Involucrar más a los interesados, para aclarar las expectativas y el valor que se espera obtener.
- Hacer uso de *checklists*, colección de métricas e inspecciones.

Metodologías Ágiles:

- Implementar técnicas para realización de pruebas, tales como TDD, *refactoring*, los cuales son usados en la metodología ágil XP y *scrum*.
- La programación en parejas en XP ayuda a reducir los defectos.
- Realizar *sprints* cortos, para incrementar la generación de entregables.

Metodologías Convencionales:

- Se debe tener buena comunicación con los interesados para recibir retroalimentación en las etapas de análisis, diseño, desarrollo y pruebas para reducir la cantidad de problemas o errores al final de la etapa, aparte de que se reducirá el riesgo.

6. Retrasos: es cuando se esperan resultados de otros procesos. Es lo contrario al concepto de *"Just in Time"*.

   Causas:

   - Cuando se espera que los interesados tomen una decisión, cuando alguien esté disponible.
   - Cuando se espera que tareas anteriores se completen.
   - Los retrasos en la espera de las aprobaciones para las solicitudes de cambio.
   - Exceso en documentación de los requerimientos.

   Recomendaciones:

   - Planificar el trabajo de manera que las personas no realicen tareas demasiado rápido y que conlleven a errores, ni que las personas tengan que esperar por resultados de otras personas que trabajan de manera lenta.
   - Se deben desglosar las actividades y procesos en taras más pequeñas para optimizar el flujo y balancear el trabajo de cada persona.
   - Se deben tener la información lo más disponible posible (*Just in Time*) para poder realizar decisiones lo más rápido posible.
   - Las interacciones y la retroalimentación ayudan a obtener información más rápidamente.
   - Se debe crear el equipo de trabajo con los conocimientos necesarios.

   Metodologías Ágiles:

   - En *scrum* se deben reportar posibles factores que creen retrasos al *Scrum Master*.
   - Se debe asegurar que los criterios de aceptación estén incluidos en cada *"user story"*
   - Se puede involucrar a los *"testers"* durante el proceso de planificación del *sprint* para reducir los defectos.

   Metodologías Convencionales:

   - Se debe definir si los requerimientos serán obtenidos por medio de documentos de requerimientos, *backlogs*, historias de usuarios, *"epics"*, etc.
   - Se necesita realizar mejoras a los procesos ya iniciados, tales como el análisis, diseño, desarrollo y pruebas.

7. La conmutación de tareas: la realización de cambios frecuentes de tareas o procesos no agregan valor y consumen recursos innecesarios. El proceso de preparar una máquina en una línea de manufactura para que realice otro proceso distinto al anterior se conoce como *"setup"* y en el caso de desarrollo de software debe ser reducido al máximo.
   Causas:
   - Interrupciones en las tareas siendo trabajadas.
   - Equipos trabajando en varios proyectos a la vez.
   - Mala coordinación de las tareas.

   Recomendaciones:
   - Se debe organizar el trabajo y los procesos para que los desarrolladores puedan enfocarse en uno el mayor tiempo posible y así aumentar la productividad.
   - Se deben diseñar los procesos de manera que las personas trabajen en tareas secuenciales en vez de paralelas.
   - No se debe asignar al personal a diferentes proyectos ya que se pierde tiempo al incorporarse, conocer el trabajo por hacer, etc.

   Metodologías Ágiles:
   - En *scrum* la idea es que los equipos sean dedicados, no deben de ser "compartidos".
   - Se debe identificar el orden de las tareas en la reunión del *sprint* para saber en cuáles tareas se va a trabajar y enfocarse en ellas para reducir el cambio de tareas.
   - Se debe reportar al *Scrum Master* acerca de interrupciones.

   Metodologías Convencionales:
   - Asignar adecuadamente a los recursos en las actividades que deben realizarse, de manera que puedan mantenerse concentrados únicamente en las asignaciones.

Como puede apreciarse, las metodologías ágiles han buscado enfatizarse en la entrega de valor a los clientes, crear productos más alineados a los criterios de aceptación, desarrollando los requerimientos solicitados, bajo una carga de trabajo aceptable por interaciones, enfatizándose siempre en el usuario. Esto permite que las actividades puedan concluirse y los cambios puedan controlarse mediante la priorización de actividades, teniendo presente la fecha de entrega y los posibles riesgos que puedan atrasar el desarrollo. De esta manera, es posible que el equipo de trabajo cuente con las condiciones necesarias para generar entregables incrementales libres de defectos y sin retrabajo por la activa participación de los interesados.

Por lo tanto, puede establecerse una mejora continua, que si se complementa con el uso de metodologías ágiles, cada iteración contará con una fase para crear ese conocimiento, que les permita incrementar la productividad y mejorar la calidad de los entregables con base en una retroalimentación continua, teniendo como meta principal del producto resultante la calidad. Para lograrlo, deben existir acciones preventivas y correctivas, para que el equipo de trabajo

pueda desempeñarse adecuadamente, evitando los defectos y la materialización de riesgos que afecten el tiempo de entrega.

El factor humano es importante en la Ingeniería de Software, dado que los líderes de los equipos de trabajo deben contar con los conocimientos y la capacidad de toma de decisiones para cumplir con los principios mencionados anteriormente. Como consecuencia, el personal como tal tiene un espacio para pensar y decidir por su cuenta sobre el ámbito donde están laborando, promoviendo una mayor participación de cada uno, una mejora continua a nivel organizacional y un mejor aseguramiento de la calidad en cada actividad que se realiza.

4. Conclusiones y Recomendaciones

De las revisiones bibliográficas efectuadas en esta investigación, en conjunto con el análisis de los casos y los resultados obtenidos, se logra determinar que para aplicar la metodología *Lean Manufacturing* en la Ingeniería de *software* (*Lean Software Development*), se recomienda tomar en cuenta los siguientes aspectos:

- Tener claras cuáles son las unidades de trabajo, como por ejemplo las Historias de Usuario.
- Las historias de usuario no pasarán a la fase de desarrollo sin antes ser aprobadas por los analistas de sistemas, arquitectos o desarrolladores.
- Las historias de usuario no pasarán a integración sin antes ser aprobadas por los *"testers"* por medio de todas las pruebas planificadas.
- El código no se liberará sin antes asegurarse de que esté libre de errores o *"bugs"* al final del proceso de integración.
- Especificar las historias de usuario como unidad de trabajo.
- Realizar seguimientos diarios.
- Realizar procesos de gestión visuales.
- Seguimiento formal a los procesos de análisis funcionales.
- Tener *"testers"* incorporados a las etapas de desarrollo.
- Congelar el código al final de cada *sprint*.
- Detener "la línea" al detectar problemas de calidad.
- Contar con los recursos e información lista (*Just in Time*) para evitar retrasos.
- Se debe reforzar la práctica de "retrospectivas" para recibir retroalimentación adecuada y sabe qué estuvo bien o mal y qué se podría mejorar en la próxima interacción, aparte que se crea una cultura de mejora continua.
- Debe fomentarse el control de procesos, la calidad fácil de apreciar, el cumplimiento, la capacidad de no detener la línea, corregir los propios errores del operador, cien por ciento verificación y la mejora continua.
- Es recomendable hacer reuniones 15 minutos por diía y una más extensa mensual.
- Los desarrolladores deben concentrarse en sus tareas, conocer los tiempos establecidos y entender sus tareas para evitar el retrabajo.
- Tener efectivas herramientas de comunicación

- Desglozar las tareas de acuerdo con la experiencia de los desarrolladores.
- Documentar y limitarse a comentar el código que se considere como de difícil comprensión.
- Cuando se cambia una línea de código debe actualizar los comentarios.
- Las pruebas no pueden ser ejecutadas por el desarrollador, ya que se corre el riesgo de ignorar escenarios de prueba críticos y producir efectos negativos en el producto.
- QA debe ejecutarse tan pronto como el proyecto inicie, mediante revisiones de lo planificado, con los requerimientos desarrollados en el *software*.
- Se recomienda hacer uso de herramientas como *Value Stram Mapping* (VSM) para conocer, de forma gráfica, el estado actual de los procesos de desarrollo de *software*, así como el construir un estado a futuro con las correcciones realizadas y el plan de mejora continua.
- Algunas otras técnicas que se podrían analizar para ser implementadas son preguntar siete veces y la técnica de las "5s".

Referencias

Leach, R. (1999). *Introduction to software engineering*. Taylor & Francis. Descargado de `https://books.google.co.cr/books?id=XW8PlP7sCOkC` pages 5

Liker, J. (2010). *Las claves del éxito de toyota: 14 principios de gestión del fabricante más grande del mundo*. Gestión 2000. Descargado de `https://books.google.co.cr/books?id=UcFoO\_bkh9YC` pages 3

Poppendieck, M. (2007). Lean software development. En *Companion to the proceedings of the 29th international conference on software engineering* (pp. 165–166). pages 1

Poppendieck, M., y Poppendieck, T. (2003). *Lean software development: An agile toolkit*. Addison-Wesley Professional. pages 3, 6

Roel, V. (1998). *La tercera revolución industrial y la era del conocimiento*. Fondo Editorial de la Universidad Nacional Mayor de San Marcos. Descargado de `https://books.google.co.cr/books?id=o9j37L4-IEsC` pages 3

Schön, E.-M., Escalona, M. J., y Thomaschewski, J. (2015). Agile values and their implementation in practice. *International Journal of Interactive Multimedia And Artificial Intelligence*, *3*(5), 61 - 66. Descargado de `http://search.ebscohost.com/login.aspx?direct=true&db=aci&AN=111492850&lang=es&site=ehost-live` pages 5

Vallon, r., Raoul1, Wenzel, L., Brüggemann, M. E., y Grechenig, T. (2015). An agile and lean process model for mobile app development: Case study into austrian industry. *Journal of Software (1796217X)*, *10*(11), 1245 - 1264. Descargado de `http://search.ebscohost.com/login.aspx?direct=true&db=aci&AN=109929711&lang=es&site=ehost-live` pages 5