**LEAN SOFTWARE DEVELOPMENT**

[**https://www.youtube.com/watch?v=dr5OPZidtNQ**](https://www.youtube.com/watch?v=dr5OPZidtNQ)

[**https://www.youtube.com/watch?v=tNIAfSWAlDE**](https://www.youtube.com/watch?v=tNIAfSWAlDE)

[**http://es.slideshare.net/jpvajda/lean-software-development-principles**](http://es.slideshare.net/jpvajda/lean-software-development-principles)

[**https://www.udemy.com/aplica-lean-desarrolla-software-just-in-time-y-con-calidad/**](https://www.udemy.com/aplica-lean-desarrolla-software-just-in-time-y-con-calidad/)

**OBJETIVOS:**

**Optimizar todo el ciclo de desarrollo de software CON PRINCIPIOS AL SENTIDO COMUN**

ENTREGAR SOFTWARE DE VALOR Y CALIDAD

SE TOMA MEJORES DECISIONES EN BASE A PRINCIPIOS

INCREMENTAR PRODUCTIVIDAD DE EQUIPO

**DEFINICION**

Este método ofrece todo un Marco Teórico Sólido y basado en la experiencia, para las prácticas ágiles de gestión. Se origino en el Sistema de Producción de Toyota y ahora, apoyado por una corriente que está surgiendo desde las métodos de desarrollo agil. La metodología de desarrollo de software lean («fino» o «esbelto») es una traducción de los principios y las prácticas de la forma de producir lean, hacia el área del desarrollo de software.

El término de desarrollo de software lean tiene origen en un libro del mismo nombre, escrito por **Mary Poppendieck y Tom Poppendieck.** El libro presenta los tradicionales principios Lean en forma modificada, así como un conjunto de 22 instrumentos y herramientas y las comparaciones con otras prácticas ágiles. La participación de Mary y Tom en la comunidad del desarrollo ágil de software, incluyendo charlas en varias conferencias, ha dado lugar a dichos conceptos, que son más ampliamente aceptados en la comunidad de desarrollo ágil. Ejemplos de ello sería *la utilización del término "lean-agile" por empresas de consultoría como NetObjectives Pace y CC, así como la inclusión de algunos de estos conceptos.*

**/Desarrollo de métodos agiles/**

Al ser un método de desarrllo  **ágil de software** envuelve un enfoque para la toma de decisiones en los proyectos de software, que se refiere a métodos de ingeniería del software basados en el [desarrollo iterativo e incremental](https://es.wikipedia.org/wiki/Desarrollo_iterativo_e_incremental), donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. Así el trabajo es realizado mediante la colaboración de equipos auto-organizados y multidisciplinarios, inmersos en un proceso compartido de toma de decisiones a corto plazo.

**Cada iteración del ciclo de vida incluye: planificación, análisis de requisitos, diseño, codificación, pruebas y documentación. Teniendo gran importancia el concepto de "Finalizado" (Done),** ya que el objetivo de cada iteración no es agregar toda la funcionalidad para justificar el lanzamiento del producto al mercado, sino incrementar el valor por medio de "software que funciona" (sin errores).

//LEAN MANUFACTURING

La **manufactura ajustada** tiene un estrecho no calculado. Es una metodología de trabajo simple, profunda y efectiva que tiene su origen en Japón, **enfocada a incrementar la eficiencia productiva en todos los procesos a partir de que se implanta la filosofía de gestión *kaizen* de mejora continua en tiempo, espacio, desperdicios, inventario y defectos involucrando al trabajador y generando en él un sentido de pertenencia al poder participar en el proceso de proponer sus ideas de cómo hacer las cosas mejor.**

***Lean manufacturing*** (‘**producción ajustada**’, ‘**manufactura esbelta**’, ‘**producción limpia**’ o ‘**producción sin desperdicios**’) es un modelo de gestión enfocado a la **creación de flujo para poder entregar el máximo valor para los clientes, utilizando para ello los mínimos recursos necesarios: es decir *ajustados*.**

La creación de flujo se focaliza en la reducción de los siete tipos de "desperdicios" en productos manufacturados:

* Sobre-producción
* Tiempo de espera
* Transporte
* Exceso de procesados
* Inventario
* Movimientos
* Defectos

Al igual se habla de un octavo desperdicio en la manufactura como lo es :

* Potencial humano subutilizado

Eliminando el despilfarro, mejora la calidad y se reducen el tiempo de producción y el costo. Las herramientas *lean* incluyen procesos continuos de análisis ( [*kaizen*](https://es.wikipedia.org/wiki/Kaizen)), producción *pull* (‘disuasión e incentivo’, en el sentido del término japonés [*kanban*](https://es.wikipedia.org/wiki/Kanban)), y elementos y procesos «a prueba de fallos» (*[poka yoke](https://es.wikipedia.org/wiki/Poka_yoke" \o "Poka yoke)*, en japonés), todo desde el [*genba*](https://es.wikipedia.org/wiki/Genba) japonés o área de valor.

Un aspecto crucial es que la mayoría de los costes se calculan en la etapa de diseño de un producto. A menudo un ingeniero especificará materiales y procesos conocidos y seguros a expensas de otros baratos y eficientes. Esto reduce los riesgos del proyecto, o lo que es lo mismo, el coste según el ingeniero, pero a base de aumentar los riesgos financieros y disminuir los beneficios. Las buenas organizaciones desarrollan y repasan listas de verificación para validar el diseño del producto.

Los principios clave del **lean manufacturing** son:

* Calidad perfecta a la primera: búsqueda de cero defectos, detección y solución de los problemas en su origen.
* Minimización del despilfarro: eliminación de todas las actividades que no son de valor añadido y redes de seguridad, optimización del uso de los recursos escasos (capital, gente y espacio).
* Mejora continua: reducción de costes, mejora de la calidad, aumento de la productividad y compartir la información.
* Procesos "pull": los productos son tirados (en el sentido de solicitados) por el cliente final, no empujados por el final de la producción.
* Flexibilidad: producir rápidamente diferentes mezclas de gran variedad de productos, sin sacrificar la eficiencia debido a volúmenes menores de producción.
* Construcción y mantenimiento de una relación a largo plazo con los proveedores tomando acuerdos para compartir el riesgo, los costes y la información.

*Lean* es básicamente todo lo concerniente a obtener las cosas correctas en el lugar correcto, en el momento correcto, en la cantidad correcta, minimizando el despilfarro, siendo flexible y estando abierto al cambio.

## Estrategia - Las 5 S

Las 5S fue un programa desarrollado por Toyota para conseguir mejoras duraderas en el nivel de organización, orden y limpieza; además de aumentar la motivación del personal.

La operatividad concreta de estos principios se instrumenta implantando una estrategia denominada y conocida internacionalmente como las 5 S por provenir de los términos japoneses:

* *seiri:* subordinar, clasificar, descartar
* *seiton:* sistematizar, ordenar
* *seiso:* sanear y limpiar
* *seiketsu:* simplificar, estandarizar y volver coherente
* *shitsuke:* sostener el proceso, disciplinar

### Seiri (subordinar, clasificar, descartar)

Es necesario iniciar en las áreas de trabajo y administrativas retirando los elementos innecesarios para la operación. Estos artículos se colocan en un lugar de almacenamiento transitorio en donde a su vez se seleccionan los que son utilizables para otra operación y se desechan o descartan los que se consideran inútiles liberando espacios y eliminando herramientas obsoletas.

### Seiton (sistematizar, ordenar)

"Un sitio para cada cosa y cada cosa en su sitio". A los elementos que no se retiraron y que se consideran necesarios se les asigna un lugar delimitando su espacio de almacenamiento, visualización, y utilización pintando líneas de señalización de áreas con líneas, siluetas, poniendo etiquetas, letreros, o utilizando muebles modulares, estantes, etc. El ordenar de esta manera otorga grandes beneficios tanto para el trabajador como para la organización

### Seiso (sanear y limpiar)

La limpieza sistematizada como parte del trabajo diario permite a su vez la inspección y la identificación de problemas de averías, desgaste, escapes o de cualquier tipo de defecto (FUGUAI) además de que da un mantenimiento regular que hace más seguro el ambiente de trabajo al disminuir los riesgos que causa la suciedad y se pueden tomar acciones concretas que reduzcan o eliminen las causas primarias de [contaminación](https://es.wikipedia.org/wiki/Contaminaci%C3%B3n) brindando como en el caso anterior beneficios directos al trabajador en su salud y seguridad así como a la organización en sí.

### Seiketsu (simplificar, estandarizar y volver coherente)

Mantener los estados de limpieza y organización utilizando los pasos anteriores. Esta etapa se puede decir que es la etapa de aplicación. Con estandarizar mantendremos permanentemente un entorno productivo e impecable, recordando los 3 principios siguientes: -Selección: No objetos innecesarios. -Orden: No desorganización. -Limpieza: No suciedad.

### Shitsuke (sostener el proceso, disciplinar)

Esta etapa es la cual mantiene que todos los pasos anteriores se cumplan paso a paso y que no se rompan los procedimientos de estos.

### Aplicación de las 5 S

Determina que el ambiente sea de calidad, es decir, que en el ambiente se puedan llevar a cabo tanto pruebas de calidad exitosas como que el producto cuente con la calidad requerida

### Objetivo de las 5 S

Lograr una mayor eficiencia, uniformidad y formalidad

### Importancia de las 5 S

Lograr la eliminación de despilfarros en diferentes áreas e incrementar el mejoramiento de condiciones de seguridad industrial

### Beneficios de las 5 S

* El empleado adquiere un sentido de pertenencia, seguridad y se siente motivado
* Se genera una cultura organizacional
* Se potencializa y se economiza el uso
* y la respuesta del tiempo
* Se incrementa la vida útil de los equipos
* Se reducen las mermas y las pérdidas por producciones con defectos
* Se elaboran productos de una mayor calidad

**//Los principios *lean***

El desarrollo *lean* puede resumirse en siete principios, similares a los principios de [manufactura esbelta](https://es.wikipedia.org/wiki/Lean_Manufacturing)/[Lean manufacturing](https://es.wikipedia.org/wiki/Lean_manufacturing)/.

**Eliminar los desperdicios**

Todo lo que no añade valor al cliente se considera un desperdicio:

* Código y funcionalidades innecesarias
* Retraso en el proceso de desarrollo de software
* Requisitos poco claros
* Burocracia
* Comunicación interna lenta

Con el fin de poder eliminar los desperdicios deberíamos ser capaces de reconocerlos y encontrarlos. Si alguna actividad podría ser excluida o el mismo resultado podría ser logrado sin ella, esta actividad es considerada un desperdicio. Los procesos y funcionalidades extra que no son usados por el cliente son desperdicios. Las esperas ocasionadas por otras actividades, equipos o procesos son desperdicio. Los defectos y la baja calidad son los desperdicios. Los gastos de gestión que no producen valor real son desperdicios. Se utiliza una técnica llamada *value stream mapping* (o mapa de flujo de valor) para distinguir y reconocer los desperdicios. El segundo paso consiste en señalar las fuentes de los desperdicios y eliminarlos. Lo mismo debe hacerse iterativamente hasta que incluso los procesos y procedimientos que parecían esenciales sean eliminados.

**Ampliar el aprendizaje**

El desarrollo de software es un proceso de aprendizaje continuo, a ello se le suman los retos de los equipos de desarrollo y el tamaño del producto final. El mejor enfoque para encarar una mejora en el ambiente de desarrollo de software es ampliar el aprendizaje. La acumulación de defectos debe evitarse ejecutando las pruebas tan pronto como el código está escrito en lugar de añadir más documentación o planificación detallada. Las distintas ideas podrían ser probadas escribiendo código e integrándolo. El proceso de recopilación de requisitos de usuarios podría simplificarse mediante la presentación de las pantallas de los usuarios finales para que estos puedan hacer sus aportes. El proceso de aprendizaje es acelerado con el uso iteraciones cortas cada una de ellas acompañada de refactorización y sus pruebas de integración.

Incrementando la retroalimentación mediante reuniones cortas con los clientes se ayuda a determinar la fase actual de desarrollo y se ajustan los esfuerzos para introducir mejoras en el futuro. Durante las reuniones, tanto los clientes como el equipo de desarrollo, logran aprender sobre el alcance del problema y buscan posibles soluciones para un mejor desarrollo. Por lo tanto, los clientes comprenden mejor sus necesidades basándose en el resultado de los esfuerzos del desarrollo y los desarrolladores aprenden a satisfacer mejor estas necesidades.

Otra idea para ampliar el aprendizaje es a través de la integración del cliente en el ambiente de desarrollo para concentrar la comunicación en las soluciones futuras y no en las soluciones posibles, promoviendo así el nacimiento de la solución a través del diálogo con el cliente.

**Decidir lo más tarde posible**

El desarrollo de software está siempre asociado con cierto grado de incertidumbre, los mejores resultados se alcanzan con un enfoque basado en opciones por lo que se pueden retrasar las decisiones tanto como sea posible hasta que éstas se basen en hechos y no en suposiciones y pronósticos inciertos. Cuanto más complejo es un proyecto, más capacidad para el cambio debe incluirse en éste, así que debe permitirse el retraso de los compromisos importantes y cruciales. El enfoque iterativo promueve este principio: la capacidad de adaptarse a los cambios y corregir los errores, ya que un error podría ser muy costoso si se descubre después de la liberación del sistema.

Un enfoque de desarrollo de software ágil puede llevarles opciones rápidamente a los clientes, lo que implica, retrasar algunas decisiones cruciales hasta que los clientes hayan reconocido mejor sus necesidades. Esto también permite la adaptación tardía a los cambios y previene las costosas decisiones delimitadas por la tecnología. Esto no significa que no haya planificación involucrada en el proceso, por el contrario, las actividades de planificación deben centrarse en las diferentes opciones y se les adapta a la situación actual; así como, se deben clarificar las situaciones confusas estableciendo las pautas para una acción rápida. Evaluar las diferentes opciones es eficaz tan pronto como queda claro que ellos no son libres, pero proporcionando la flexibilidad necesaria para una tardía toma de decisiones.

**Reaccionar tan rápido como sea posible**

En la era de la rápida evolución tecnológica, no es el más grande quien sobrevive, sino el más rápido. Cuanto antes se entrega el producto final sin defectos considerables más pronto se pueden recibir comentarios y se incorporan en la siguiente iteración. Cuanto más cortas sean las iteraciones, mejor es el aprendizaje y la comunicación dentro del equipo. Sin velocidad, las decisiones no pueden ser postergadas. La velocidad asegura el cumplimiento de las necesidades actuales del cliente y no lo que éste requería para ayer. Esto les da la oportunidad de demorarse pensando lo que realmente necesitan, hasta que adquieran un mejor conocimiento. Los clientes valoran la entrega rápida de un producto de calidad.

La ideología de producción [*Just In Time*](https://es.wikipedia.org/wiki/Just_In_Time) podría aplicarse a programas de desarrollo, reconociendo sus necesidades específicas y el ambiente. Lo anterior se logra mediante la presentación de resultados , la necesidad de dejar que el equipo se organice y dividiendo las tareas para lograr el resultado necesario para una iteración específica.

Al principio, el cliente dispone los requisitos necesarios. Esto podría ser simplemente presentar los requisitos en pequeñas fichas o historias y los desarrolladores estimarán el tiempo necesario para la aplicación de cada tarjeta. Así, la organización del trabajo cambia en sistema autopropulsado: cada mañana durante una reunión inicial cada miembro del equipo evalúa lo que se ha hecho ayer, lo que hay que hacer hoy y mañana y pregunta por cualquier nueva entrada necesaria de parte de sus colegas o del cliente. Esto requiere la transparencia del proceso, que es también beneficioso para la comunicación del equipo.

Otra idea clave del Sistema de Desarrollo de Producto de la Toyota se establece a base de diseño. Si un nuevo sistema de frenos es necesario para un coche, por ejemplo, tres equipos pueden diseñar soluciones al mismo problema. Cada equipo aprende sobre el ambiente del problema y diseños de una posible solución. Cuando una solución se considera irrazonable, se desecha. Al final de un periodo, los diseños sobrevivientes se comparan y se elige uno, quizá con algunas modificaciones basadas en el aprendizaje de los demás, un gran ejemplo de compromiso aplazado hasta el último momento posible. Las decisiones en el software también podrían beneficiarse de esta práctica para minimizar el riesgo provocado por un solo gran diseño realizado por adelantado.

**Potenciar el equipo**

Ha habido una creencia tradicional en la mayoría de las empresas acerca de la toma de decisiones en la organización: los administradores dicen a los trabajadores cómo hacer su propio trabajo. En una técnica llamada Work-Out, los roles cambian: a los directivos se les enseña a escuchar a los desarrolladores, de manera que éstos puedan explicar mejor qué acciones podrían tomarse, así como ofrecer sugerencias para mejoras. Los directores de proyecto más experimentados simplemente han declarado la clave de éxito de los proyectos: "Buscar la buena gente y dejarles hacer su propio trabajo".

Otra creencia errónea ha sido considerar a las personas como recursos. Las personas podrían ser los recursos desde el punto de vista de una hoja de datos estadísticos, pero en el desarrollo de software, así como cualquier organización de negocios, las personas necesitan algo más que la lista de tareas y la seguridad de que no será alterada durante la realización de las tareas. Las personas necesitan motivación y un propósito superior para el cual trabajar: un objetivo alcanzable dentro de la realidad con la garantía de que el equipo puede elegir sus propios compromisos. Los desarrolladores deberían tener acceso a los clientes; el jefe de equipo debe proporcionar apoyo y ayuda en situaciones difíciles, así como asegurarse de que el escepticismo no arruine el espíritu de equipo.

**Crear la integridad**

El siempre exigente cliente debe tener una experiencia general del sistema. A esto se le llama percepción de integridad: ¿Cómo es publicitado, entregado, implementado y accedido? ¿Es su uso intuitivo? ¿Precio? ¿Hasta qué punto resuelve los problemas?. Integridad Conceptual significa que los componentes separados del sistema funcionan bien juntos, como en un todo, logrando equilibrio entre la flexibilidad, sostenibilidad, eficiencia y capacidad de respuesta. Esto podría lograrse mediante la comprensión del dominio del problema y resolviéndolo al mismo tiempo, no secuencialmente. La información necesaria es recibida por los pequeños lotes, no en una vasta cantidad y con una preferible comunicación cara a cara, sin ninguna documentación por escrito. El flujo de información debe ser constante en ambas direcciones, a partir del cliente a los desarrolladores y viceversa, evitando así la gran y estresante cantidad de información después de un largo periodo de desarrollo en el aislamiento.

Una de las maneras más saludables hacia una arquitectura integrante es la refactorización. Cuantas más funcionalidades se añaden a las del sistema, mas se pierde del código base para futuras mejoras. Así como en la [Programación extrema](https://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema) (XP), la [refactorización](https://es.wikipedia.org/wiki/Refactorizaci%C3%B3n) es mantener la sencillez, la claridad, la cantidad mínima de funcionalidades en el código. Las repeticiones en el código son signo de un mal diseño de código y deben evitarse. El completo y automatizado proceso de construcción debe ir acompañada de una suite completa y automatizada de pruebas, tanto para desarrolladores y clientes que tengan la misma versión, sincronización y semántica que el sistema actual. Al final, la integridad debe ser verificada con una prueba global, garantizando que el sistema hace lo que el cliente espera que haga. Las pruebas automatizadas también son consideradas como parte del proceso de producción y, por tanto, si no agregan valor deben considerarse residuos. Las pruebas automatizadas no deberían ser un objetivo, sino, un medio para un fin; específicamente para la reducción de defectos.

**Véase todo el conjunto**

Los sistemas de software hoy en día no son simplemente la suma de sus partes, sino también el producto de sus interacciones. Los defectos en el software tienden a acumularse durante el proceso de desarrollo por medio de la descomposición de las grandes tareas en pequeñas tareas y de la normalización de las diferentes etapas de desarrollo. Las causas reales de los defectos deben ser encontradas y eliminadas. Cuanto más grande sea el sistema, más serán las organizaciones que participan en su desarrollo y más partes son las desarrolladas por diferentes equipos y mayor es la importancia de tener bien definidas las relaciones entre los diferentes proveedores con el fin de producir una buena interacción entre los componentes del sistema.

La manera de pensar ofrecida Lean tiene que ser bien entendida por todos los miembros de un proyecto antes de aplicarlo de manera concreta en una situación de la vida real.

"Piensa en grande, actúa en pequeño, equivócate rápido; aprende con rapidez" estas consignas resumen la importancia de comprender el terreno y la idoneidad de implementar los principios Lean lo largo del proceso de desarrollo de software. Sólo cuando todos los principios de Lean se aplican al mismo tiempo, combinado con un fuerte "sentido común" en relación con el ambiente de trabajo, hay una base para el éxito en el desarrollo de software.

**//LEAN DEVELOPMENT**

Por David J. Anderson. David J. Anderson es el autor de tres libros, Lessons en administración de Agile: En el Road a Kanban, que se publicó en 2012, Kanban: Cambio correcto de Evolutionary del negocio de Tecnología, [1] que se publicó en 2010, y, Administración de Agile para el software Engineering: Aplicando la teoría de las restricciones del negocio Resultados, [2] que se publicó en 2003. Era un miembro del equipo que creó el método de desarrollo del Software Agile, desarrollo basado en características, en Singapur entre 1997 y 1999. Se creó MSF for CMMI process Improvement, y se co-creó la nota técnica del software Engineering Institute, “CMMI y Agile: Porqué no abrazo ambos!” Éste es un fundador de sistemas Society de Lean (<http://www.leansystemssociety.org>). Es CEO de la Universidad Lean-Kanban Inc., una organización de entrenamiento acreditada y estándares de calidad que ofrece training en Kanban a través de una red de socios en el mundo y lidera una firma internacional de consultoría y entrenamiento en Gestión, David J. Anderson & Associates Inc. (<http://www.agilemanagement.net>) que ayuda a las empresas de tecnología a mejorar su rendimiento con mejores políticas de administración y toma de decisiones.

En noviembre de 2011, noviembre de 2012 actualizado.

Anderson describe el sistema Lean Software Development.

## [Se aplica a](javascript:void(0))

Lean, desarrollo de software, administración de proyectos, Team Foundation Server

* [Introduction](https://msdn.microsoft.com/es-es/library/hh533841%28v=vs.120%29.aspx#BKMK_intro)
* [Lean y Agile](https://msdn.microsoft.com/es-es/library/hh533841%28v=vs.120%29.aspx#BKMK_LeanAgile)
* [Lean más allá de Agile](https://msdn.microsoft.com/es-es/library/hh533841%28v=vs.120%29.aspx#BKMK_leanbeyondagile)
* [Definir Lean Software Development](https://msdn.microsoft.com/es-es/library/hh533841%28v=vs.120%29.aspx#BKMK_DefiningLeanSftwareDev)
* [Valores](https://msdn.microsoft.com/es-es/library/hh533841%28v=vs.120%29.aspx#BKMK_values)
* [Principios](https://msdn.microsoft.com/es-es/library/hh533841%28v=vs.120%29.aspx#BKMK_principles)
* [Procedimientos](https://msdn.microsoft.com/es-es/library/hh533841%28v=vs.120%29.aspx#BKMK_practices)

El término Lean Software Development (desarrollo de software ajustado) se acuñó por primera vez como título de una conferencia organizada por la iniciativa ESPRIT de la Unión Europea, en Stuttgart, Alemania, en octubre de 1992. Independientemente, al año siguiente, Robert “Bob” Charette en 1993, planteó el concepto de “Lean Software Development” como parte de su trabajo para investigar mejores formas para administrar los riesgos en proyectos de software. El término “Lean” (ajustado) data de 1991, y fue sugerido por James Womack, Daniel Jones y Daniel Roos en su libro La máquina que cambió el mundo[3] como el término en inglés para describir el método de administración utilizado en Toyota. La idea que Lean podría aplicarse al desarrollo de software se estableció muy temprano, solo 1 o 2 años después de que el término se utilizara asociado a tendencias en procesos de fabricación y en ingeniería industrial.

En su segundo libro, publicado en 1995, Womack y Jones[4] definieron cinco pilares básicos del patrón de pensamiento Lean. Estos eran:

* Valor
* Flujo de valor
* Flujo
* Extracción
* Perfección

Esta sería la definición operativa predeterminada para la metodología Lean durante la mayor parte de la década siguiente. Se sugirió que la búsqueda de la perfección se consiguió eliminando residuos. Aunque había 5 pilares, fue el quinto, la búsqueda de la perfección a través de la identificación sistémica de actividades generadoras de residuos y su eliminación, la que resonó realmente en gran parte de la audiencia. Lean se asoció prácticamente de forma exclusiva a la práctica de eliminación de desechos desde finales de los años 90 hasta comienzos del siglo XXI.

La definición de Womack y de Jones para Lean no se admite universalmente. Los principios de administración en Toyota son mucho más sutiles. La palabra inglesa “waste” (residuos) se describe de una manera más precisa mediante tres términos en japonés:

* Muda: literalmente indica “desecho” pero implicando actividad sin valor añadido.
* Mura: indica “desigualdad” y se interpreta como “variabilidad de flujo”
* Muri: significado “sobrecarga” o “irracional”

La perfección se persigue a través de la reducción de la actividades sin valor añadido pero también a través de perfeccionar el flujo y eliminado la sobrecarga. Además, el enfoque de Toyota se basaba en un respecto fundacional por las personas que estaba muy influenciado por las enseñanzas de control de calidad del siglo XX y por expertos en control de procesos estadísticos como W. Edwards Deming, destacados representantes del ámbito de control de calidad del siglo XX.

Desgraciadamente, hay casi tantas definiciones de Lean como autores sobre el tema.

## [Lean y Agile](javascript:void(0))

Invitaron a Bob Charette pero no pudo asistir a la reunión de 2001 en Snowbird, Utah, donde se creó el Manifiesto para el desarrollo del Software Agile[5]. A pesar de faltar a esta reunión histórica, Lean Software Development fue considerado como uno de varios enfoques de Agile respecto al desarrollo de software. Jim Highsmith dedicó un capítulo de su libro de 2002[6] a una entrevista con Bob sobre el tema. Más adelante, Mary y Tom Poppendieck continuaron con una serie de 3[7,8,9] libros. Durante los primeros años del siglo XXI, los principios de Lean se usaron para explicar por qué los métodos del software Agile eran mejores. Según la metodología Lean los métodos de Agile contenían pocos “desechos” y por consiguiente generaron mejores resultados económicos. Los principios de Lean se usaron como un “otorgante de permisos” para adoptar los métodos de Agile.

## [Lean más allá de Agile](javascript:void(0))

En años recientes, Lean Software Development ha emergido realmente como una disciplina única relacionada con el movimiento Agile, pero sin ser específicamente un subconjunto del mismo. Esta evolución se inició con la síntesis de las ideas de desarrollo de productos según la metodología Lean y el trabajo de Donald G. Reinertsen[10,11] y las ideas procedentes del mundo no ágil de ingeniería de sistemas a gran escala y de los escritos de James Sutton y Peter Middleton[12]. También sinteticé el trabajo de Eli Goldratt y W. Edwards Deming desarrolló un enfoque sobre el flujo en lugar de una reducción de desechos[13]. En instancias de Reinertsen hacia 2005, introduje el uso de los sistemas kanban que restringen el trabajo en curso y “extraen” nuevo trabajo solo cuando el sistema está listo para procesarlo. Alan Shalloway sumó sus pensamientos en el desarrollo de software Lean en su libro de 2009 sobre el tema[14]. Desde 2007, la aparición de Lean como nueva fuerza en el progreso de la profesión de desarrollo de software se ha centrado en mejorar el flujo, la administración del riesgo y la toma de decisiones (de administración). Kanban se ha convertido en un habilitador para las iniciativas de Lean en trabajo relativo a TI. Parece que centrarse en el flujo, en lugar de en la eliminación de desechos, está probando ser mejor catalizador para la mejora continuada en actividades de trabajos del conocimiento, como el desarrollo de software.

## [Definir Lean Software Development](javascript:void(0))

La definición de Lean Software Development es un reto porque no hay ningún método o proceso específico de Lean Software Development. Lean no es un equivalente de Personal Software Process, V-Model, Spiral Model, EVO, Feature-Driven Development, Extreme Programming, Scrum o Test-Driven Development. Un proceso del ciclo de vida de desarrollo de software o un proceso de administración de proyectos podría decirse que es "Lean" si se comprobado que cumple los valores del movimiento Lean Software Development y los principios del Lean Software Development. Los que anticipen una receta simple que se puede seguir y denominar Lean Software Development estarán decepcionados. Debe crear o ajustar su propio proceso de desarrollo de software comprendiendo los principios de Lean y adoptando los valores básicos de Lean.

Hay varias escuelas de pensamiento de Lean Software Development. El mayor, y discutible el espacio, la escolar es sistemas Society de Leer, que incluye Donald Reinertsen, Jim Sutton, Alan Shalloway, Bob Charette, Mary Poppendeick, y j de David. Anderson. El trabajo de Mary y de Tom Poppendieck desarrollado antes de la formación de Society y su credo se coloca por separado, como el trabajo de Craig Larman, de Bas Vodde [15,16], y, recientemente, de Jim Coplien [17]. Búsquedas de este artículo de ser ampliamente representante del punto de vista de los sistemas Society de Eficientes como se expresa en su credo y proporcionar una síntesis y un resumen de las ideas.

## [Valores](javascript:void(0))

Los sistemas Society de Lean publicados su credo [18] en el software 2012 de Lean y la [19] conferencia de sistemas. Esto se basaba en un conjunto de valores publican un año anterior. Esa los valores incluyen:

* Aceptar la condición humana
* Aceptar que la complejidad y la incertidumbre son naturales para el trabajo de conocimiento
* Trabaje con vistas hacia un mejor resultado económico
* Mientras se posibilita un mejor resultado de sociológico
* Buscar, adoptar y cuestionar ideas de una amplia gama de disciplinas
* Una comunidad basada en valores mejora la velocidad y la profundidad de cambios positivos

### [Aceptar la condición humana](javascript:void(0))

El trabajo de conocimiento como el desarrollo de software lo realizan personas. Los seres humanos somos inherentemente complejos y, aunque somos pensadores lógicos, también nos dejamos llevar por nuestras emociones y algunos rasgos animales inherentes que no se pueden dominar tan fácilmente. Nuestra psicología y neuropsicología debe tenerse en cuenta cuando se diseñan los sistemas o procesos en los que trabajamos. El comportamiento social también debe incorporarse. Los seres humanos son por naturaleza emocionales, sociales y tribales, y nuestro comportamiento cambian con el cansancio y la tensión. Los procesos correctos serán los que adopten e implementen la condición humana, no los que tratan de denegarla y asumir un comportamiento lógico propio de máquinas.

### [Aceptar que la complejidad y la incertidumbre son naturales para el trabajo de conocimiento](javascript:void(0))

El comportamiento de los clientes y los mercados es imprevisible. El flujo de trabajo a lo largo de un proceso y una colección de trabajadores es imprevisible. Los defectos y el trabajo por duplicado necesario son imprevisibles. El desarrollo de software conlleva riesgos o comportamientos aparentemente aleatorios en muchos niveles. El propósito, los objetivos y el ámbito de los proyectos suelen cambiar mientras se entregan. Parte de esta incertidumbre y variabilidad, aunque inicialmente desconocida, se puede conocer en el sentido de que se puede estudiar y cuantificar, así como administrar sus riesgos, pero cierta variabilidad es desconocida de antemano y no se puede anticipar de forma adecuada. Como resultado, los sistemas de Lean Software Development deben poder responder a los eventos unfold y el sistema debe poder adaptarse las circunstancias fluctuantes. Por consiguiente, cualquier proceso de Lean Software Development debe existir dentro de un marco que permite la adaptación (del proceso) para eventos unfold.

### [Trabaje con vistas hacia un mejor resultado económico](javascript:void(0))

Las actividades humanas como Lean Software Development se deben centrar en generar un mejor resultado económico. El capitalismo es aceptable cuando contribuye al valor empresarial y beneficia al cliente. Los inversores y los propietarios de negocios merecen una mayor rentabilidad de la inversión. Los empleados y los trabajadores merecen una tarifa justa por su esfuerzo en la realización del trabajo. Los clientes merecen un buen producto o servicio que ofrezca las ventajas prometidas a cambio de un precio justo. Mejores resultados económicos implican una entrega de más valor al cliente, a menor precio, mientras que se administra el capital implementado por parte de los inversores o los propietarios de la manera más eficaz posible.

### [Habilitar un mejor resultado sociológico](javascript:void(0))

No se deben obtener mejores resultados económicos a expensas de los que están realizando el trabajo. Es esencial crear un lugar de trabajo que respete a personas mediante la aceptación de la condición humana y que proporcione sistemas de trabajo que respeten la naturaleza psicológica y sociológica de las personas. Crear un buen sitio para hacer un gran trabajo es un valor básico para la comunidad de Lean Software Development

## [Principios](javascript:void(0))

La comunidad Lean Software & Systems parece estar de acuerdo con algunos principios que respaldan los procesos de Lean Software Development.

* Siga Systems Thinking & Design Approach
* Los resultados emergentes pueden estar influenciados por la creación de la arquitectura del contexto para un sistema adaptable complejo
* Respetar a las personas (como parte del sistema)
* Utilice el modo científico (para controlar las mejoras)
* Fomentar el liderazgo
* Genere visibilidad (en los trabajos, flujos de trabajo y operación del sistema)
* Reducir el tiempo de flujo
* Reducir residuos para mejorar la eficacia

### [Siga Systems Thinking & Design Approach](javascript:void(0))

Esto se conoce a menudo en la literatura de Lean como “optimizar el todo”, lo que implica que es el resultado de todo el sistema (o el proceso) lo que se desea optimizar, y no se deben optimizar partes con la esperanza equivocada de que esto optimizará el conjunto mágicamente. La mayoría de los usuarios creen que la consecuencia es así, que la optimización de elementos (optimización local) dará lugar a un resultado que no será óptimo.

Lean Systems Thinking and Design Approach requiere tener en cuenta las peticiones en el sistema realizadas por las partes interesadas externas, como clientes, y el resultado deseado requerido por tales partes interesadas. Debemos estudiar la naturaleza de la demanda y compararla con la capacidad de entrega de nuestro sistema. La petición incluirá la denominada “petición de valor”, por la que los clientes están dispuestos a pagar, y la “petición de errores”, que normalmente implica una revisión o una petición adicional a causa de un error en el suministro de la de petición de valor. Las peticiones erróneas tiene dos formas por lo general: revisión del trabajo respecto a una petición de valor previamente satisfecha y servicios adicionales o soporte debido a un error a la hora de entregar la petición de valor. En el desarrollo de software, la demanda de error suele solicitar una corrección de errores así como una función de atención al cliente o de soporte técnico.

Un enfoque de diseño de sistemas requiere que también sigamos el enfoque de planear, hacer, estudiar, actuar (Plan-Do-Study-Act, PDSA) para procesar el diseño y las mejoras. Hora Edwards Deming usó las palabras “estudiar” y “capacidad” para indicar que estudiamos la filosofía natural del comportamiento del sistema. Este sistema se compone de nuestro proceso de desarrollo de software y de todas las personas involucradas en él. Tendrá un comportamiento visible en términos de plazos de entrega, calidad, cantidad de las características o funciones entregadas (lo cual se denominado en la literatura de Agile “velocidad "), etc. Estas métricas mostrarán variabilidad y, estudiando la media y la dispersión de las variaciones, podemos entender mejor nuestra capacidad. Si no se corresponden correctamente con la demanda y las expectativas del cliente, el sistema necesitará ser rediseñado para cerrar la brecha.

Deming también nos enseñó que la capacidad está influenciada en un 95% por el diseño del sistema y solo el 5% está condicionado por el rendimiento de las personas. Es decir, podemos respetar a personas no culpándolas por un desequilibrio entre capacidad y la demanda y rediseñando el sistema para permitirles hacer bien las cosas.

Para entender diseño de sistemas, debemos tener conocimientos científicos de la dinámica de la capacidad del sistema y cómo puede verse afectada. Los modelos se desarrollan para predecir la dinámica del sistema. Cuando hay muchos modelos posibles, se suelen usar los más populares: la comprensión de costos económicos; los denominados costos de transacción y de coordinación relacionados con la producción de productos o servicios valorados por el cliente; la teoría de las restricciones (la comprensión de los cuellos de botella); y la teoría del conocimiento profundo (el estudio y el reconocimiento de si las variaciones son propias del diseño del sistema, o especiales y externas al diseño del sistema).

### [Los resultados emergentes pueden estar influenciados por la creación de la arquitectura del contexto para un sistema adaptable complejo](javascript:void(0))

Los sistemas complejos tienen condiciones de inicio y reglas simples que, cuando se ejecutan en iteración, generan un resultado emergente. Los resultados emergentes son difíciles o imposibles de predecir, dadas las condiciones de inicio. El experimento informático “The Game of Life” es un ejemplo de un sistema complejo. Un sistema adaptable complejo incluye cierta conciencia y un método interno de reflexión que le permita tener en cuenta en qué medida su conjunto de reglas actual le permite lograr los resultados deseados. El sistema adaptable complejo podrá entonces optar por adaptarse él mismo (para cambiar sus reglas simples) para cerrar el hueco entre el resultado actual y el resultado deseado. El juego de la vida adaptado de modo que las reglas se puedan volver a escribir durante el juego sería un sistema adaptable complejo.

En los procesos de desarrollo de software, las “reglas sencillas” de sistemas adaptables complejos son las directivas que constituyen la definición de proceso. El principio básico aquí se basa en la creencia de que desarrollar productos y servicios de software no es una actividad determinista, y por consiguiente un proceso definido que no puede adaptarse a sí mismo no será una respuesta adecuada a eventos imprevistos. Por consiguiente, el proceso diseñado como parte de nuestro patrón de pensamiento y enfoque del diseño debe ser personalizable. Se adapta a través de la modificación de las directivas de las que se compone.

El enfoque de Kanban a Lean Software Development utiliza este concepto tratando las directivas del sistema de extracción kanban como “las reglas sencillas”, y las condiciones iniciales son que el trabajo y el flujo de trabajo se visualizan, el flujo se administra mediante la comprensión del la dinámica del sistema y la organización utiliza un enfoque científico para comprender, proponer e implementar mejoras de procesos.

### [Respetar a las personas](javascript:void(0))

La comunidad Lean adopta la definición de Peter Drucker de trabajo de conocimiento que afirma que los trabajadores son trabajadores de conocimiento si están más informados sobre el trabajo que realizan que sus jefes. Esto crea la implicación de que los trabajadores están en una posición mejor para tomar decisiones sobre cómo realizar el trabajo y cómo modificar los procesos para mejorar los métodos de trabajo. Por lo que la voz del trabajador debe ser respetada. Los trabajadores deben estar facultados para organizarse por su cuenta para completar el trabajo y lograr los resultados deseados. También deben tener autorización para sugerir e implementar oportunidades de mejora de procesos o “eventos de kaizen”, como se conocen en la literatura Lean. Crear directivas de proceso explícitas de modo que los trabajadores sean conscientes de reglas que restringen su actividad es una forma de respetarles. Las reglas bien definidas fomentan una organización autónoma ya que eliminan el miedo y la necesidad de tener valor. Respetar a las personas dándoles autoridad y ofreciéndolas un conjunto de directivas declaradas explícitamente permite ser fiel al valor básico de respeto a la condición humana.

### [Use el método científico](javascript:void(0))

Utilice modelos para entender la dinámica de cómo se realiza el trabajo y de cómo funciona el sistema Lean Software Development. Observe y estudie el sistema y su capacidad y, a continuación, desarrolle y aplique los modelos para predecir su comportamiento. Recopile datos cuantitativos en las revisiones y use esos datos para entender cómo rinde el sistema y predecir cómo puede cambiar cuando cambia el proceso.

La comunidad Lean Software & Systems utiliza métodos estadísticos, como gráficos de control de proceso estadístico e histogramas de análisis espectral de datos sin formato para conseguir tiempo y velocidad para comprender la capacidad del sistema. También utilizan modelos como: la teoría de las restricciones para entender los cuellos de botella; el sistema de conocimiento profundo para entender la variación inherente al diseño del sistema frente a la que procede de influencias externas; y un análisis de costos económicos para las tareas meramente de coordinación, configuración, entrega o limpieza posteriores a la creación de productos o servicios valorados por el cliente. Otros modelos se están empezando a utilizar, como la teoría de la opción de real, que intenta aplicar teoría de la opción financiera de la administración de riesgos financieros a la toma de decisiones del mundo real.

El método científico sugiere: estudiamos; postulamos un resultado en función de un modelo; perturbamos el sistema basándonos en esa predicción; y observamos de nuevo para ver si la perturbación produjo los resultados que el modelo predijo. De no ser así, comprobamos los datos y reconsideramos si nuestro modelo es exacto. El uso de modelos para controlar las mejoras de procesos lo convierte en una actividad científica, dejando de ser una actividad supersticiosa basada en la intuición.

### [Fomentar el liderazgo](javascript:void(0))

La dirección y administración no son iguales. Administración es la actividad de diseñar procesos, crear, modificar y eliminar directivas, de tomar decisiones estratégicas y operativas, de recopilar recursos, de proporcionar financiación e instalaciones y de comunicar información sobre contexto como estrategia, objetivos y resultados deseados. La dirección tiene que ver con la visión, la estrategia, la táctica, el valor, la innovación, la valoración, la propugnación y muchos otros atributos. La dirección puede y debe proceder de cualquier persona dentro de una organización. Pequeños actos de liderazgo de los trabajadores crearán una cascada de mejoras que posibilitarán los cambios necesarios para crear un proceso de Lean Software Development.

### [Genere visibilidad](javascript:void(0))

El trabajo de conocimiento no es visible. Si no puede ver algo, es (casi) imposible administrarlo. Es necesario generar visibilidad en el trabajo que se emprende y en el flujo de ese trabajo a través de una red de personas, conocimientos y departamento hasta que se complete. Es necesario crear visibilidad en el diseño del proceso encontrando maneras de visualizar el flujo del proceso y creando directivas del proceso explícitas para que todo el mundo las vea y las tenga en cuenta. Cuando todas estas cosas estén visibles, el uso del método científico es posible, y las conversaciones sobre posibles mejoras podrán ser colaborativas y objetivas. La mejora del proceso de colaboración es casi imposible si el trabajo y el flujo de trabajo no están visibles y si las directivas de proceso no son explícitas.

### [Reducir el tiempo de flujo](javascript:void(0))

Los profesionales del desarrollo de software y el personal académico que estudia ingeniería de software tradicionalmente se han centrado en medir el tiempo empleado en trabajar en una actividad. La comunidad de Lean Software Development ha descubierto que podría ser más útil medir el tiempo transcurrido real que algo tarda en procesarse. Esto se conoce normalmente como duración del ciclo y normalmente se califica mediante los límites de las actividades realizadas. Por ejemplo, la duración del ciclo hasta llegar a Listo para la implementación mediría el tiempo total transcurrido para un elemento de trabajo, como un caso de usuario, que debe analizarse, diseñarse, desarrollarse, probarse en varios sentidos, y colocarse en cola listo para su implementación en un entorno de producción.

El centrarse en el tiempo que se necesita para recorrer el proceso es importante de varias maneras. Se han mostrado duraciones de ciclo más largas que se correlacionan con un crecimiento no lineal en el porcentaje de errores. Por consiguiente, los tiempos del ciclo llevan a una mayor calidad. Esto es contraintuitivo, ya que parece ridículo que los errores se puedan insertar en el código mientras está esperando en la cola y realmente no lo está tocando nadie. Tradicionalmente, la profesión de ingeniería del software y los académicos que la han estudiado han omitido este tiempo de inactividad. Sin embargo, las pruebas empírica sugieren que la duración de ciclo es importante para la calidad inicial.

Alan Shalloway también habla sobre el concepto de “trabajo inducido”. Su observación es que un retraso en realizar una tarea puede conducir a que esa tarea que requiera muchos más esfuerzos de los necesarios. Por ejemplo, encontrar y corregir inmediatamente un error puede llevar tan solo 20 minutos, pero si ese error se evalúa, se pone en la cola y después se esperan varios días o semanas a que se solucione, podría llevar muchas horas su corrección. Por consiguiente, el retraso en el tiempo del ciclo tiene trabajo adicional “inducido”. Como este trabajo es evitable, en términos de Lean, debe considerarse como un “desperdicio”.

La tercera razón para centrarse en la duración del ciclo es una razón relacionada con el negocio. Cada característica, función o caso de usuario tiene un valor. Ese valor puede ser incierto pero, sin embargo, hay un valor. El valor puede variar con el tiempo. El concepto de valor variable con el tiempo se puede expresar económicamente como una función de rentabilidad de mercado. Cuando se comprende la función de rentabilidad de mercado para un elemento de trabajo, aunque la función presente una variedad de valores para modelar la incertidumbre, es posible evaluar un “costo de retraso”. El costo de retraso permite poner un valor en la reducción del tiempo del ciclo.

Con algunos elementos de trabajo, la función de rentabilidad de mercado no se inicia hasta una fecha futura conocida. Por ejemplo, una característica diseñada para utilizarse durante el día festivo del 4 de julio en los Estados Unidos de América no tiene valor antes de esa fecha. Acortar el tiempo de ciclo y poder predecir su duración con cierta confiabilidad sigue siendo útil en este ejemplo. En el mejor de los casos, queremos empezar con el trabajo para entregar la característica “en el plazo exacto” cuando sea necesaria y no significativamente antes de la fecha deseada, ni tarde, ya que una entrega fuera del plazo conlleva costos por retraso. La entrega a su debido tiempo garantiza que el uso óptimo se creó con recursos disponibles. Una entrega antes de tiempo implica que puede que hayamos trabajado en algo más y, implícitamente, haber incurrido en un costo de oportunidad por aplazamiento.

Como resultado de estas tres razones, Lean Software Development busca minimizar el tiempo de flujo y grabar los datos que permitan realizar predicciones sobre el tiempo de flujo. El objetivo es minimizar la demanda por errores debida a errores, los residuos de la sobrecarga debida al retraso en la solución de errores y maximizar el valor evitando los costos por retraso y los costos de oportunidad por aplazamiento.

### [Reducir residuos para mejorar la eficacia](javascript:void(0))

Para cada actividad que agregue valor, hay actividades de configuración, de limpieza y de entrega que son necesarias pero no agregan valor por sí mismas. Por ejemplo, una iteración del proyecto que desarrolla un incremento de software que funciona requiere un planeamiento (una actividad de configuración), un entorno y quizás una bifurcación de código en el control de versiones (conocido en su conjunto como administración de configuración y también como actividad de configuración), un plan de lanzamiento y realizar el lanzamiento en sí (una actividad de entrega), una demostración al cliente (una actividad de entrega) y posiblemente destruir o reconfigurar el entorno (una actividad de limpieza.) En términos económicos, las actividades de configuración, limpieza y entrega son costos de transacción para llevar a cabo el trabajo de valor añadido. Estos costos (o gastos generales) se consideran residuos en la metodología Lean.

Cualquier forma de sobrecarga de comunicación puede considerarse un desperdicio. Las reuniones para determinar el estado del proyecto y para programar o asignar trabajo a miembros del equipo se considerará un costo de coordinación en terminología económica. Todos los costos de coordinación son irrelevantes en esta percepción de Lean. Los métodos de Lean Software Development pretenden eliminar o reducir los costos de coordinación con la colocación de miembros del equipo, reuniones cara a cara de poca duración como reuniones sin sentarse y controles visuales como los muros de cartas.

El tercer tipo común de residuos del Lean Software Development es la demanda por errores. Las peticiones erróneas son una carga para sistema de desarrollo de software. Las peticiones de erróneas suelen conllevar un trabajo de revisión u otras formas de trabajo como efecto secundario de una baja calidad. Las formas más habituales de demanda por errores en el desarrollo de software son los errores, los defectos de producción y las actividades de soporte al cliente derivadas de un uso incorrecto del software. El porcentaje de trabajo en curso que es demanda por errores suele denominarse carga por errores. El porcentaje de trabajo de valor añadido frente a la demanda por errores es una medida de la eficacia del sistema.

El porcentaje de trabajo de valor añadido frente al trabajo total, incluidos todos los costos de coordinación y transacción que no añaden valor, determina el nivel de eficacia. Un sistema sin costes de transacción y de coordinación y las cargas erróneas no se considerarán eficaces al 100%.

Tradicionalmente, las ciencias empresariales occidentales han enseñado que la eficacia se puede mejorar aumentando el tamaño de lote de trabajo. Normalmente, los costos de transacción y coordinación son fijos o suben ligeramente con un aumento del tamaño de lote. Como resultado, los lotes grandes de trabajo son más eficaces. Este concepto se conoce como “economía de escala”. Sin embargo, con los problemas relacionados con los trabajos de conocimiento, los costos de coordinación tienden a subir de forma no lineal con el tamaño del lote, mientras que los costos de transacción puede presentar a menudo un crecimiento lineal. Como resultado, el enfoque tradicional del siglo XX relativo a la eficacia no es adecuado para los problemas de trabajo sobre los conocimientos el desarrollo de software.

Es mejor centrarse en la reducción de las sobrecargas conservando un tamaño pequeño de tamaño de lote para mejorar la eficacia. Por consiguiente, la forma según Lean de ser eficaz es reducir los desechos. Los métodos de Lean Software Development se centran en métodos rápidos, baratos y de planeación rápida; sobrecarga baja de comunicación; y mecanismos de coordinación de sobrecargas bajas efectivos, como controles visuales en sistemas kanban. También promueven la automatización de las pruebas y de la implementación para reducir los costos de transacción de las entregas. Las herramientas modernas para minimizar los costos de configuración y de destrucción de entorno, como sistemas de control de versiones modernos y uso de la virtualización, también ayudan a mejorar la eficacia de pequeños lotes de desarrollo de software.

## [Procedimientos](javascript:void(0))

Lean Software Development no recomienda procedimientos. Es más importante mostrar que las definiciones de proceso reales están en concordancia con los principios y valores. Sin embargo, varios procedimientos se están adoptando normalmente. Esta sección proporciona información general breve sobre algunas de ellas.

### [Diagramas de flujo acumulativos](javascript:void(0))

Los diagramas de flujo acumulativos llevan siendo una parte estándar de la elaboración de informes en Team Foundation Server desde 2005. Los diagramas de flujo acumulativos trazan un gráfico de área de los elementos de trabajo acumulativos en cada estado de un flujo de trabajo. Contienen una gran cantidad de información y pueden utilizarse para derivar la duración de ciclo media entre los pasos de un proceso, así como la tasa de rendimiento (o “velocidad”). Diversos procesos del ciclo de vida de desarrollo de software generan diferentes signaturas visuales en diagramas de flujo acumulativos. Los usuarios pueden reconocer los patrones del mal funcionamiento en el proceso que se muestra en el gráfico de áreas. Un proceso verdaderamente Lean mostrará true áreas uniformemente distribuidas de color, incrementándose naturalmente a un ritmo constante. La imagen aparecerá suavizada sin pasos escalonados o bloques visibles de color.

En su forma más básica, los diagramas de flujo acumulativos se usan para visualizar la cantidad de trabajo en curso en un momento determinado en el ciclo de vida del elemento de trabajo. Esto se puede utilizar para detectar cuellos de botella y para observar los efectos de “mura” (variabilidad de flujo).

### [Controles visuales](javascript:void(0))

Además de usar diagramas de flujo acumulativos, los equipos de Lean Software Development usan las placas físicas, o proyecciones de sistemas electrónicos de visualización, para visualizar el trabajo y observar el flujo. Estas visualizaciones ayudan a los miembros del equipo a observar la acumulación de trabajo en curso y les permite ver los cuellos de botella y los efectos de “mura”. Los controles visuales también permiten a los miembros del equipo organizarse entre ellos para elegir el trabajo y colaborar juntos sin una planeación o una dirección o intervención específica de administración. Estos controles visuales se denominan a menudo “muros de cartas” o a veces (incorrectamente) “paneles kanban”.

### [Sistemas Kanban virtuales](javascript:void(0))

Un sistema kanban es una práctica adoptada de la fabricación Lean. Usa un sistema de cartas físicas para limitar la cantidad de trabajo en curso en cualquier fase determinada en el flujo de trabajo. Tales sistemas limitados de trabajo en curso crean una “extracción” donde se inicia el nuevo trabajo solo cuando hay kanban libres que indiquen que dicho trabajo se puede “extraer” en un estado determinado y el trabajo puede progresar en él.

En Lean Software Development, el kanban es virtual y su seguimiento se realiza normalmente estableciendo un número máximo para un paso determinado en el flujo de trabajo de un tipo de elemento de trabajo. En algunas implementaciones, los sistemas electrónicos llevan un seguimiento del kanban virtual y proporcionan una señal cuando el trabajo se puede iniciar. La señal puede ser visual o en forma de alerta, como un correo electrónico.

Los sistemas kanban virtuales se combinan con los controles visuales para proporcionar un sistema kanban virtual visual que representa el flujo de trabajo de uno o varios tipos de elemento de trabajo. Tales sistemas se denominan a menudo “paneles kanban” o “sistemas kanban electrónicos”. Un sistema kanban virtual visual, denominado Visual WIP[20], está disponible como complemento para Team Foundation Server,. Este proyecto lo desarrolló Hakan Forss en Suecia como código abierto.

### [Tamaños pequeños de lote/flujo de parte única](javascript:void(0))

Lean Software Development requiere que el trabajo se realice en pequeños lotes, lo que a menudo se llama “iteraciones” o “incrementos” o los elementos de trabajo fluyan de forma independiente, lo que se denomina “flujo de la parte única”. El flujo de la parte única requiere una estrategia de administración de la configuración sofisticada del trabajo completado que va a entregarse mientras que el trabajo no terminado no se lanza accidentalmente. Esto normalmente se logra utilizando estrategias de bifurcación en el sistema de control de versiones. Un lote de trabajo pequeño normalmente se considera un lote que puede asumir un equipo pequeño de 8 personas o menos en 2 semanas.

Los lotes pequeños y el flujo de parte única requieren interacción frecuente con los propietarios del negocio para reponer el trabajo pendiente, la cola o el trabajo. También requieren cierta capacidad de lanzar versiones con frecuencia. Para permitir la interacción frecuente con los propietarios del negocio y las entregas frecuentes, es necesario reducir los costos de transacción y de coordinación de ambas actividades. Una manera común de conseguirlo es usar la automatización.

### [Automatización](javascript:void(0))

Lean Software Development espera un nivel alto de automatización para habilitar de forma económica el flujo de la parte única y fomentar la alta calidad y reducir la demanda errónea. El uso de la prueba automatizada, la implementación automatizada y los generadores de software para automatizar la implementación de los modelos de diseño y la creación de secciones repetitivas de poca variabilidad de código fuente serán habituales en los procesos Lean Software Development.

### [Eventos de Kaizen](javascript:void(0))

En la literatura de Lean, el término kaizen quiere decir “mejora continua” y un evento de kaizen es el acto de crear un cambio en un proceso o herramienta que se supone que desembocará en una mejora.

Los procesos de Lean Software Development usan diversas actividades para generar eventos kaizen. Se enumeran aquí. Cada una de estas actividades se ha diseñado para estimular una conversación sobre los problemas que afectan negativamente a la capacidad y, por consiguiente, a la capacidad cumplir los requisitos de la demanda. La esencia de kaizen en el trabajo de conocimiento es que debemos provocar conversaciones sobre problemas entre grupos de usuarios de equipos diferentes y con distintos conocimientos.

### [Reuniones informales diarias](javascript:void(0))

Los equipos de desarrolladores de software, a menudo formados por hasta 50 personas, se agruparán normalmente delante de un sistema de control visual como una pizarra que muestra una visualización del trabajo en curso. Discuten la dinámica del flujo y los factores que afectan al flujo de trabajo. Se presta especial atención al trabajo externamente bloqueado y al trabajo con retrasos a causa de errores. Los problemas con el proceso por lo general se hacen evidentes tras varias reuniones informales. El resultado es que un grupo más pequeño puede quedarse tras la reunión para analizar el problema y proponer una solución o un cambio en los procesos. Luego se producirá un evento de kaizen. Estas reuniones espontáneas a menudo se conocen como círculos de calidad espontáneos en la literatura más antigua. Este tipo de reuniones espontáneas son el corazón de una cultura realmente de kaizen. Los administradores fomentarán la aparición de eventos kaizen después de reuniones informales diarias para controlar la adopción de Lean en la organización.

### [Retrospectivas](javascript:void(0))

Los equipos de proyecto pueden programar reuniones regulares para reflejar el rendimiento reciente. Estas suelen tener lugar después de completar determinadas entregas del proyecto o después de los incrementos de tiempo de desarrollo denominados iteraciones o sprints en Agile Software Development.

Las retrospectivas utilizan normalmente un enfoque anecdótico para la reflexión haciendo preguntas como “¿qué salió bien?", “¿qué haríamos de forma diferente?” y “¿qué debemos dejar de hacer?”

Las retrospectivas normalmente generan un trabajo pendiente de sugerencias para eventos de kaizen. El equipo puede dar prioridad a algunas de ellas para la implementación.

### [Revisiones de operaciones](javascript:void(0))

Una revisión de las operaciones normalmente es mayor que una retrospectiva e incluye representantes de una secuencia completa de valores. Es común que hasta 12 departamentos presenten datos objetivos y cuantitativos que muestren la demanda obtenida y que reflejen su capacidad de satisfacer esta demanda. Las revisiones de operaciones por lo general se efectúan mensualmente. Las diferencias clave entre una revisión de operaciones y una retrospectiva es que las revisiones de operaciones abarcan un conjunto más amplio de funciones, generalmente una cartera de proyectos y otras iniciativas, y utilizan datos objetivos y cuantitativos. Las retrospectivas, por el contrario, suelen restringirse a un proyecto único; implican a pocos equipos, por ejemplo análisis, desarrollo y prueba; y son, por lo general, de naturaleza anecdótica.

Una revisión de las operaciones provocará debates sobre la dinámica que afecta al rendimiento entre equipos. Quizás un equipo genera una petición errónea que procesa otro equipo. Quizás esa demanda errónea es molesta y provoca que un segundo equipo no cumpla sus compromisos y no entregue según las expectativas. Una revisión de las operaciones proporciona una oportunidad de explicar tales problemas y de proponer cambios. Las revisiones de operaciones por lo general generan un pequeño trabajo pendiente de eventos kaizen potenciales que a los que se pueden asignar prioridades y programar para la implementación posterior.

No existe un único proceso de Lean Software Development. Se puede decir que un proceso es Lean si se corresponden claramente con los valores y los principios de Lean Software Development. Lean Software Development no recomienda ningún procedimiento, pero algunas actividades se han convertido en habituales. Las organizaciones que trabajan con Lean tratan de fomentar el kaizen a través de la visualización del flujo de trabajo y de los trabajos en curso y a través de una comprensión de la dinámica de flujo y de factores (como cuellos de botella, disponibilidad no inmediata y desechos) que le afectan. Las mejoras en el proceso se sugieren y se justifican como una forma de reducir las fuentes de variabilidad, eliminar desechos, mejorar los flujos o mejorar la entrega de valor o la administración de riesgos. Como tal, los procesos de Lean Software Development evolucionarán siempre y estarán individualizados de forma única para la organización dentro en la que se desarrollan. No se puede copiar una definición de proceso desde una organización a otra y pretender que funcione en un contexto diferente. También es improbable que si se regresa a una organización después de unas semanas o meses, el proceso en uso sea igual al anteriormente observado. Evolucionará siempre.

Una organización que utilice un proceso de desarrollo de Lean se puede decir que es Lean si solo exhibió cantidades pequeñas de residuos en sus tres formas (“mura”, “muri” y “muda") y se puede demostrar que optimiza la entrega de valor mediante una administración efectiva del riesgo. La búsqueda de la perfección en Lean siempre es un viaje. No hay ningún destino. Las organizaciones que trabajan con Lean buscan siempre mejoras adicionales.

Lean Software Development sigue siendo un campo emergente y se puede esperar que continúe evolucionando durante la década próxima.

1 2 3 4 5

2 1 3 4 5