Universidad de Valladolid

E. U. de Informática (Segovia)

Ingeniería Técnica en Informática de Gestión

Metodología LEAN para el desarrollo de software. Ejemplo práctico de aplicación en empresa de desarrollo de software

Alumno: María Isabel Sanz Mateo

Tutor: José Vicente Álvarez Bravo

Metodología LEAN para el desarrollo de software.
Ejemplo práctico de aplicación en empresa de desarrollo de software

Contenido

0. PRESENTACIÓN, OBJETIVOS Y ALCANCE

En la última década, si bien la preocupación por la reducción de costes y la mejora de la productividad ha sido una constante, la explosión de la reciente crisis mundial la ha convertido en la tónica general. En el difícil contexto económico actual, donde uno de los principales factores de riesgo es la contracción del consumo de productos y demanda de servicios, la filosofía Lean y su visión de la producción toma protagonismo, especialmente si tenemos en cuenta que, en su origen, dio respuesta con éxito a una situación adversa de características similares. Prueba de ello es que la empresa que creó esta filosofía y su principal estandarte en la actualidad continúa con su liderazgo en el sector y demuestra su capacidad de reacción en plena crisis:

El grupo automovilístico japonés Toyota registró un beneficio neto de 190.4 mil millones de yenes durante el primer trimestre del año fiscal 2010-2011 (de abril a junio), en comparación con las pérdidas de 77.8 mil millones de yenes contabilizadas en el mismo periodo de 2009.

La filosofía Lean se fundamenta en dar un protagonismo especial al componente humano de la producción y en una declarada vocación de mejora continua. Cuestiones como la visión a largo plazo, el esfuerzo por afrontar nuevos retos, el espíritu de mejora continua, el respeto por las personas y el trabajo en equipo son términos que sientan las bases sobre la que se construye el éxito de esta filosofía. Desde sus orígenes, estas ideas se han ido perfilando en la forma de una serie de principios para llevarlas a cabo y que en 2001 acabarán plasmándose en el llamado “Manual de Estilo Toyota”. Basadas en estos principios, existe un conjunto de técnicas y prácticas, algunas de ellas bastante extendidas y aplicadas en multitud de disciplinas. La aplicación de éstas continúa extendiéndose, así como nuevas prácticas van surgiendo para dar respuesta a nuevos problemas, sin apartarse del camino trazado por los principios de esta filosofía.

En cuanto al enfoque con que se trata el proceso de producción en particular, gira en torno a tres ideas básicas:

- Fabricar únicamente lo que se necesita.
- Eliminar aquello que no añada valor al producto.
- Detener la producción si algo va mal y corregir la fuente del error.

Dada la efectividad que demuestra esta filosofía en aquellas disciplinas donde se encuentra asentado, está emergiendo el interés por la adaptación de esta filosofía para la mejora de servicios y productos “intelectuales”, entre los cuales se encuentra la Ingeniería del Software.

Esta disciplina tiene como objetivo fundamental dar soporte a otras áreas de negocio y prácticamente está presente de manera más o menos directa en todo aquello que nos rodea. Su importancia se pone de manifiesto porque dada su naturaleza “de apoyo” a sus resultados, para bien o para mal, inciden directamente en los resultados del ámbito al que se aplique y en la actualidad parece aceptado que el empleo de las TIC (Tecnologías de la Información y Comunicaciones) jugará un papel fundamental en la mejora de la productividad y competitividad de las empresas para salir de la crisis.

El objetivo de este proyecto es aportar una visión general de las metodologías denominadas ágiles centrándonos en la metodología LEAN. Esta metodología tiene de especial que nace fuera del marco del desarrollo del software con el fin de desarrollar una cultura hacia una organización más eficiente mediante los cambios en el proceso de negocio, y que como veremos es totalmente aplicable al desarrollo de software.
Más que una metodología, se la puede considerar una filosofía de trabajo, aplicable a cualquier gestión empresarial.
En este trabajo se pretende plasmar cómo puede cambiar la gestión del desarrollo de proyectos software dentro de un entorno real al pasar a usar una metodología ágil como es Lean. El entorno real consiste en uno de los principales bancos españoles y con gran presencia internacional. En esta empresa, se está implantando la metodología Lean no sólo en sus departamentos de desarrollo de software, si no como filosofía de gestión de toda la compañía.

Intentaré reflejar, desde mi experiencia profesional, cómo nos ha afectado este cambio en el desarrollo de nuevos proyectos software dentro de esta organización.

El contenido se ha estructurado en 8 apartados principales:

Tras esta presentación del proyecto fin de carrera, se dispone de un apartado de introducción al concepto de ingeniería del software donde se verá la problemática inherente a esta disciplina y a las necesidades surgidas para el nacimiento de un nuevo tipo de metodologías, las metodologías ágiles. En el siguiente apartado se centra en mostrar un resumen de la aparición de las metodologías ágiles en el desarrollo del software.
Este proyecto pretende mostrar la aplicación de la metodología Lean. Al ser ésta una metodología no específica del desarrollo de software, se hace una exposición de los principales conceptos, principios y técnicas relacionados con la filosofía Lean y para los que se ha encontrado referencia en el ámbito de la Ingeniería del Software. El ánimo de este apartado es dar una visión más o menos somera de cada uno de los elementos en su concepción original para facilitar la comprensión de su correspondencia en el apartado dedicado al Lean aplicado a la Ingeniería del Software. A continuación se destacan las referencias más significativas encontradas de Lean aplicado a la Ingeniería del Software y algunas aplicaciones prácticas, siguiendo un esquema similar al creado en el apartado de filosofía Lean para facilitar su lectura.

Finalmente se pasa a la exposición de la implantación práctica de la metodología Lean dentro de una organización con sus correspondientes conclusiones.

1. INTRODUCCIÓN

¿Qué se entiende por Software?

"Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación."

¿Qué se entiende por Ingeniería del Software?

"La aplicación práctica, sistemática, disciplinada y cuantificable del conocimiento científico para realizar el análisis de las necesidades del usuario y obtener el software y la documentación asociada requerida para su desarrollo, operación y mantenimiento de manera rentable, fiable, certificada y que opere en máquinas reales."

El proceso de desarrollo es intensamente intelectual y se ve afectado por la creatividad y juicio de las personas involucradas en el mismo. Aunque un proyecto de desarrollo de software es equiparable en muchos aspectos a cualquier otro proyecto de ingeniería, en el desarrollo de software hay una serie de desafíos adicionales relativos a la naturaleza del producto a obtener, como son la intangibilidad o la fiabilidad del mismo.

1.1 Problemática de la Ingeniería del Software

¿Qué complejidad inherente tiene el proceso de desarrollo de Software?

El proceso de desarrollo es intensamente intelectual y se ve afectado por la creatividad y juicio de las personas involucradas en el mismo. Aunque un proyecto de desarrollo de software es equiparable en muchos aspectos a cualquier otro proyecto de ingeniería, en el desarrollo de software hay una serie de desafíos adicionales relativos a la naturaleza del producto a obtener, como son la intangibilidad o la fiabilidad del mismo:

Intangibilidad

Hablamos de un producto intangible y normalmente complejo, dado que su cometido es dar respuesta a la abstracción de un problema planteado por personas que normalmente desconocen esta disciplina. Por esta cuestión, definir con exactitud los requisitos a cubrir y consolidarlos tempranamente se complica con frecuencia, haciendo inevitable el cambio, durante el desarrollo o una vez acabado el mismo.

Fiabilidad

Un producto software en sí es complejo siendo inviable conseguir el 100% de fiabilidad en un programa por muy reducida que sea su funcionalidad. Esto ocurre por la elevada combinatoria asociada a los distintos factores que intervienen en la ejecución del mismo y que impiden una verificación de las todas posibles situaciones que se puedan presentar, son ejemplo de estos factores:

- Datos introducidos por el usuario
- Datos almacenados en el sistema
- Interacción con el software base o sistema operativo Interacción o el hardware del sistema sobre el que se ejecuta
- Interacción con otras aplicaciones
- Etc.

Se definen un conjunto de actividades aplicables a cualquier proyecto, una serie de conjuntos de tareas que permitirán la adaptación de las actividades a cada proyecto en particular y requisitos del equipo de proyecto y un conjunto de actividades de protección, independientes de cualquier actividad del marco de trabajo, presentes durante todo el proceso y destinadas a cuestiones como la garantía de calidad, gestión de la configuración, gestión de riesgos, etc. De una forma más concreta, las actividades del marco de trabajo se pueden asimilar a tres fases genéricas:

- **Definición**: Destinada identificar qué se pretende obtener. Establecer cuestiones como qué información debe procesar el sistema, qué funcionalidad implementar, cómo comportarse, etc. Durante esta fase tendrán lugar tareas como la ingeniería de sistemas o de información, planificación de proyecto y análisis de requisitos.
- **Desarrollo**: en la que efectivamente se da solución a las necesidades definidas en la fase anterior, y su cometido principal es la construcción del producto final. En esta fase tienen lugar tareas como el diseño, la codificación y pruebas del software.
- **Mantenimiento**: que se ocupa del cambio del producto obtenido en la fase de desarrollo. Este cambio puede venir motivado por distintas cuestiones, distinguiéndose fundamentalmente cuatro tipo de cambios:

    - Correcciones de defectos localizados tras el desarrollo, cuestión inevitable, como explicamos con anterioridad. Las modificaciones realizadas para dar solución a estos cambios se conocen como mantenimiento correctivo.
    - Adaptaciones del software ante el cambio del entorno original para el que fue desarrollado, cambios del hardware, del software base, etc. Realizadas por el mantenimiento adaptativo.
    - Mejoras que den respuesta a nuevos requisitos del cliente, cubiertas por el mantenimiento perfectivo
    - Prevención de la degradación que se produce por el cambio. El mantenimiento preventivo o reingeniería del software se ocupa de realizar cambios que faciliten la corrección, adaptación y mejora.

Metodologías

Las metodologías deben dar forma y detalle al proceso de desarrollo software y, adicionalmente, proporcionar un conjunto de prácticas y técnicas recomendadas, así como guías de adaptación a los distintos proyectos. Habitualmente suelen ser combinación de modelos de proceso genéricos.

Para el ámbito de este trabajo podemos dividirlas en:

- Metodologías tradicionales
- Metodologías ágiles

Teniendo en cuenta la filosofía de desarrollo de las metodologías, aquellas con mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, reciben el apelativo de Metodologías Tradicionales o Pesadas. Estas metodologías tradicionales imponen una disciplina de trabajo sobre el proceso de desarrollo del software, con el fin de conseguir un software más eficiente. Para ello, se hace énfasis en la planificación total de todo el trabajo a realizar y una vez que está todo detallado, comienza el ciclo de desarrollo del producto software. Se centran especialmente en el control del proceso, mediante una rigurosa definición de roles, actividades, artefactos, herramientas y notaciones para el modelado y documentación detallada.
Además, las metodologías tradicionales no se adaptan adecuadamente a los cambios, por lo que no son métodos adecuados cuando se trabaja en un entorno, donde los requisitos no pueden predecirse o bien pueden variar.

2. METODOLOGÍAS ÁGILES

2.1. Marco histórico

En el siglo XIX Charles Babbage concibió y diseño la “máquina analítica”, la primera máquina de propósito general, la cual podía ser programada e implementaba el concepto de almacenamiento de datos, bucles e instrucciones de decisión, de modo que el programa se ejecutaba sin la intervención de una persona que la operase. Aunque esta máquina puede considerarse, conceptualmente, como el primer computador, en realidad nunca se construyó. Por tal razón, muchos historiadores atribuyen el calificativo de “primer computador” al Z3, construido en Alemania por Konrad Zuse en los 40`s, la cual era digital, multiusos y, a diferencia de los dos primeros modelos (Z1 y Z2), plenamente funcional. De manera simultánea en Inglaterra y Estados Unidos se adelantaron proyectos de investigación que años más tarde dieron como resultado máquinas más sofisticadas y más veloces, programables, de propósitos generales y capaces de ejecutar un considerable número de tareas de mayor complejidad.

En la siguiente década, la programación paso de ser externa (conexión de cables y dispositivos) a ser almacenada en la memoria del computador; estos primeros programas se caracterizaron por un alto grado de dependencia entre los lenguajes, el computador y las personas que los escribían. Para enfrentar tal situación surgieron los lenguajes de alto nivel, los compiladores y los programas almacenados en medios magnéticos, haciendo el proceso de elaboración de programas más exigente, dadas las numerosas y nuevas posibilidades respecto al uso del computador. Sin embargo, en una disciplina naciente la falta de técnicas o metodologías, hicieron evidente el exceso en los tiempos empleados, los sobrecostos, la insatisfacción de los usuarios o clientes; en definitiva, la baja calidad de os programas. Esto fue lo que finalmente desembocó en la llamada crisis del software. Como respuesta a esta crisis, en la segunda mitad de la década de los 60 se popularizó el concepto de ingeniería de software. Al formalizarse una metodología para la elaboración de programas bajo conceptos de ingeniería, esta tarea se transformó en un proceso riguroso propendiendo por el cumplimiento de los cronogramas, los presupuestos, y lo más importante, la satisfacción del cliente.

La cronología de la aparición de las distintas metodologías de desarrollo se pueden clasificar en:

1. Pre – metodologías: entre los años 50 y 60.
2. Primeras metodologías: entre los años 1970 y 1980.
3. La era de las metodologías: entre los años 80 y los primeros años de los 90.
4. Era post-metodológica: En los años 90.

La era pre metodológica se caracteriza por la ausencia de metodologías, grandes computadoras, las cuales ejecutaban aplicaciones científicas y militares.
Poco a poco se fueron introduciendo en organizaciones empresariales realizando por ejemplo informes de ventas, guardando información de usuarios o archivos. Los sistemas eran desarrollados por programadores, planificados como pequeños ejercicios, se ejecutaban en cortos plazos de tiempo. Los usuarios no siempre quedaban satisfechos con las soluciones presentadas, la documentación era escasa y a veces no existía, los cambios requerían más tiempo y complejizaban la aplicación, y los programadores eran

cotizados porque eran los únicos que conocían el funcionamiento. Se dieron cuenta que necesitaban más tiempo para el análisis y diseño de las aplicaciones, aparecieron las figuras de los analistas de sistemas quienes eran los intermediarios entre el programador y el sistema; y, los operadores que controlaban el funcionamiento de las computadoras.

La era de las primeras metodologías comienza en 1971 con el nacimiento del modelo en cascada (Royce, 1970). Esta metodología consiste en una serie de etapas que es necesario 5 cumplir secuencialmente, cada etapa tiene asociada un producto entregable para ser considerada finalizada. Entre los inconvenientes que poseía este modelo se encuentran la imposibilidad de obtener todas las necesidades de los clientes, la inestabilidad, la inflexibilidad, la insatisfacción de los clientes, exceso de documentación y la facilidad de desviarse de los tiempos planificados.
En la era de las metodologías surgieron un gran número de nuevos enfoques en respuesta a las limitaciones del modelo en cascada. Se pueden mencionar dos corrientes: la que busca mejorar el modelo en cascada y la que decide hacer algo diferente. Asimismo, en los 70 surgieron una variedad de técnicas y herramientas que mejoraron el modelo, tales como: el modelo entidad-relación, la normalización, diagramas de flujos de datos, diagramas de actividad, diagramas estructurados; y herramientas de gestión de proyectos de software, repositorios de código, de modelado, y las Computer Aided Software Engineering (CASE). Entre las metodologías que actualizan o mejoran el modelo en cascada se pueden encontrar: MERISE, Structured Systems Analysis and Design Method o Yourdon System Method. Estas integran las herramientas y técnicas mencionadas.

Entre las metodologías alternativas emergieron:

- Sistemas. Utilizan técnicas cualitativas para aplicar pensamiento sistemático a situaciones no sistemáticas, un ejemplo es Soft Systems Methodologies.
- Estratégicas. Se enfocan en la planificación previa al desarrollo de software y obtener los objetivos de negocio, un ejemplo es Business Systems Planning.
- Participativas. Se caracterizan por la participación de los usuarios y demás interesados en las fases de análisis, diseño e implementación, un ejemplo es ETHICS.
- Prototipadas. Se distinguen por el uso de prototipos que permite a los usuarios visualizar y validar el desarrollo, un ejemplo es Rapid Application Development.
- Estructuradas. Extienden los conceptos de programación estructurada al análisis y diseño, con técnicas que permiten el análisis descendente y representación compleja de procesos.
- Análisis de datos. Se centra en entender y documentar los datos, la técnica ampliamente utilizada es el modelado entidad – relación, un ejemplo es Information Engineering.

Los primeros años de la década del 90 nacieron un nuevo grupo de metodologías:

- Orientadas a objetos. Enfocada en objetos que representan entidades del mundo real.

- Desarrollo incremental o evolutivo. Enfocado en prototipado y desarrollo de software incremental. La metodología Dynamic System Development Methods.
- Específicas. Fueron concebidas para un tipo de proyectos, tienen un propósito determinado:

    - Welti, para el desarrollo de aplicaciones Enterprise Resource Planning.
    - CommonKADS, para el desarrollo de aplicaciones de gestión del conocimiento.
    - Process Innovation6 para el desarrollo de aplicaciones de reingeniería de procesos de negocio.
    - Renaissance, metodología que se basa en la técnica de ingeniería inversa para sistemas heredados.
    - Web IS Development Methodology, para el desarrollo de aplicaciones basadas en el Web.

A fines de los 90 trasciende una corriente caracterizada por una seria reevaluación de los supuestos beneficios de las metodologías, reacciones casi violentas que dan lugar a la era postmetodológica. Las alternativas que se presentaron en esta era fueron:

- Desarrollo externo. Muchas empresas decidieron no realizar más proyectos software y compraban todos sus requisitos empaquetados.
- Mejora continua. Abogan por la continua mejora de las metodologías.
- Desarrollo ad hoc y contingencia. El desarrollo ad hoc es un regreso a la era premetodológica, sin metodología.
- Desarrollo contingente o flexible. Es un enfoque más suavizado, menos prescriptivo, donde existe una estructura pero sus componentes se dejan a elección de los desarrolladores.
- Desarrollo ágil. Se enfoca en la participación de los usuarios y clientes más que en procesos y herramientas, trabajando más en el software y menos en la documentación, colaborando más con clientes en vez de negociar y responder a los cambios por encima de la planificación de los tiempos del trabajo.

2.2. La necesidad de nuevos métodos

Hasta hace poco en el proceso de desarrollo se hacía mucho énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos, incluyendo modelado y documentación detallada. Este esquema "tradicional" para abordar el desarrollo de software ha demostrado ser efectivo y necesario en proyectos de gran tamaño (respecto a tiempo y recursos), donde por lo general se exige un alto grado de ceremonia en el proceso. Sin embargo, este enfoque no resulta ser el más adecuado para muchos de los proyectos actuales donde el entorno del sistema es muy cambiante, y en donde se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad. Ante las dificultades para utilizar metodologías tradicionales con estas restricciones de tiempo y flexibilidad, muchos equipos de desarrollo se resignan a prescindir del “buen hacer” de la ingeniería del software, asumiendo el riesgo que ello conlleva. En este escenario, las metodologías ágiles emergen como una posible respuesta para llenar ese vacío metodológico.

Para tener una visión global y comprender el origen y la razón de ser de las metodologías ágiles hay que remontarse a 1.968, momento en que se le puso nombre a los continuos retrasos, sobrecostes y deficiencias de calidad o utilidad que se producían en el desarrollo de software denominándolo como “crisis del software”. Un problema que se producía asiduamente y que había que solucionar. La solución que se buscó pasaba por:

El desarrollo de una ingeniería del software. Un cuerpo científico de conocimiento y prácticas especializado.

La aplicación de la gestión predictiva (clásica) de los proyectos. Basada en la planificación del trabajo, su ejecución y posterior control con el claro objetivo de garantizar el cumplimiento de lo planificado en términos de tiempo, coste y calidad.

Y la producción basada en procesos. El objetivo es garantizar la calidad del resultado final a través del uso en la producción de unos procesos de calidad.

La realidad fue que la gestión de proyectos predictiva no funcionaba como debiera en entornos inestables, con gran competencia y cuando los requisitos iniciales cambiaban. Entonces, en 1.986, Takeuchi & Nonaka publicaron un artículo llamado “The New New Product Development Game” donde daban ejemplos de empresas punteras que estaban obteniendo buenos resultados en ese tipo de entornos y bajo esas circunstancias aplicando metodologías que contravenían la hasta entonces forma de gestionar proyectos. Se empezó a vislumbrar el nacimiento del “agilismo” que surge como antítesis de los modelos de desarrollo basados en procesos.

Será en 2.001 en una reunión celebrada en Salt Lake City donde se debatiría sobre desarrollo de software cuando se acuña el término “Métodos Ágiles” y surgen los cuatro principios que conforman el “Manifiesto Ágil”, base de este “agilismo” y de métodos de gestión, tales como: DSDM, Scrum, XP, FDD, …

Estos nuevos métodos de desarrollo surgen como respuesta a un nuevo entorno cambiante, muy competitivo, donde los lanzamientos de productos y las mejoras son

cada vez más continuos y se producen en menores intervalos de tiempo y el valor en alza, que otorga la ventaja competitiva para estar en los primeros puestos de un sector, es la innovación.

Autores como el mencionado Ries, además apuntan a que esa innovación ha de producirse en el menor tiempo posible y con el menor esfuerzo necesario, otorgando a la variable tiempo y a la inversión racionalizada una importancia primordial.

2.3. El manifiesto Ágil

Las metodologías ágiles surgen por ser las más adecuadas para proyectos pequeños donde el entorno del sistema es muy cambiante y se exige reducir al máximo los tiempos de desarrollo manteniendo una alta calidad.

Como comentamos en el punto anterior, el 17 de febrero de 2001 diecisiete críticos de los modelos de mejora del desarrollo de software basados en procesos, convocados porKent Beck, quien había publicado un par de años antes Extreme Programming Explained, libro en el que exponía una nueva metodología denominada Extreme Programming, se reunieron enSnowbird, Utah para tratar sobre técnicas y procesos para desarrollar software. En la reunión se acuñó el término "Métodos Ágiles" para definir a los métodos que estaban surgiendo como alternativa a las metodologías formales (CMMI, SPICE) a las que consideraban excesivamente "pesadas" y rígidas por su carácter normativo y fuerte dependencia de planificaciones detalladas previas al desarrollo.

Los integrantes de la reunión resumieron los principios sobre los que se basan los métodos alternativos en cuatro postulados, lo que ha quedado denominado como Manifiesto Ágil y que podemos encontrar en la web http://www.agilemanifesto.org/iso/es/manifesto.html

Hasta 2005 han sido frecuentes las posturas radicales entre los defensores de los modelos de procesos y los defensores de modelos ágiles, quizá más ocupados en descalificar al otro que en estudiar sus métodos y conocerlos para mejorar los propios.
En el Manifiesto Ágil, firmado por Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert Cecil Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland y Dave Thomas, se expone que:

> *Estamos poniendo al descubierto mejores métodos para desarrollar software, haciéndolo y ayudando a otros a que lo hagan. Con este trabajo hemos llegado a valorar:*
>
> - *A los **individuos y su interacción**, por encima de los procesos y las herramientas.*
> - *El **software que funciona**, por encima de la documentación exhaustiva.*
> - *La **colaboración con el cliente**, por encima de la negociación contractual.*
> - *La **respuesta al cambio**, por encima del seguimiento de un plan.*

Aunque hay valor en los elementos de la derecha, valoramos más los de la izquierda.

2.3.1. Valores del Manifiesto Ágil

I. Valorar más a los individuos y su interacción que a los procesos y las herramientas.

Este es posiblemente el principio más importante del manifiesto. Por supuesto que los procesos ayudan al trabajo. Son una guía de operación. Las herramientas mejoran la eficiencia, pero sin personas con conocimiento técnico y actitud adecuada, no producen resultados.

Las empresas suelen predicar muy alto que sus empleados son lo más importante, pero la realidad es que en los años 90 la teoría de producción basada en procesos, la reingeniería de procesos ha dado a estos más relevancia de la que pueden tener en tareas que deben gran parte de su valor al conocimiento y al talento de las personas que las realizan.

Los procesos deben ser una ayuda y un soporte para guiar el trabajo. Deben adaptarse a la organización, a los equipos y a las personas; y no al revés. La defensa a ultranza de los procesos lleva a postular que con ellos se pueden conseguir resultados extraordinarios con personas mediocres, y lo cierto es que este principio es peligroso cuando los trabajos necesitan creatividad e innovación.

II. Valorar más el software que funciona que la documentación exhaustiva

Poder ver anticipadamente cómo se comportan las funcionalidades esperadas sobre prototipos o sobre las partes ya elaboradas del sistema final ofrece una retroalimentación muy estimulante y enriquecedor que genera ideas imposibles de concebir en un primer momento; difícilmente se podrá conseguir un documento que contenga requisitos detallados antes de comenzar el proyecto.

El manifiesto no afirma que no hagan falta. Los documentos son soporte de la documentación, permiten la transferencia del conocimiento, registran información histórica, y en muchas cuestiones legales o normativas son obligatorios, pero se resalta que son menos importantes que los productos que funcionan. Menos trascendentales para aportar valor al producto.

Los documentos no pueden sustituir, ni pueden ofrecer la riqueza y generación de valor que se logra con la comunicación directa entre las personas y a través de la interacción con los prototipos. Por eso, siempre que sea posible debe preferirse, y reducir al mínimo indispensable el uso de documentación, que genera trabajo que no aporta un valor directo al producto.

Si la organización y los equipos se comunican a través de documentos, además de perder la riqueza que da la interacción con el producto, estos documentos se acaban empleando de forma defensiva como barricadas ante departamentos o personas.

III. Valorar más la colaboración con el cliente que la negociación contractual

Las prácticas ágiles están especialmente indicadas para productos difíciles de definir con detalle en el principio, o que si se definieran así tendrían al final menos valor que si se van enriqueciendo con retro-información continua durante el desarrollo. También para los casos en los que los requisitos van a ser muy inestables por la velocidad del entorno de negocio.

Para el desarrollo ágil el valor del resultado no es consecuencia de haber controlado una ejecución conforme a procesos, sino de haber sido implementado directamente sobre el producto. Un contrato no aporta valor al producto. Es una formalidad que establece líneas divisorias entre responsabilidades, que fija los referentes para posibles disputas contractuales entre cliente y proveedor.

En el desarrollo ágil el cliente es un miembro más del equipo, que se integra y colabora en el grupo de trabajo. Los modelos de contrato por obra no encajan.

IV. Valorar más la respuesta al cambio que el seguimiento de un plan

Para un modelo de desarrollo que surge de entornos inestables, que tienen como factor inherente el cambio y la evolución rápida y continua, resulta mucho más valiosa la capacidad de respuesta que la de seguimiento y aseguramiento de planes pre-establecidos. Los principales valores de la gestión ágil son la anticipación y la adaptación; diferentes a los de la gestión de proyectos ortodoxa: planificación y control para evitar desviaciones sobre el plan.

2.4. Los 12 principios ágiles

Bajo el concepto de principio se hace referencia a las características que hacen la diferencian entre un proceso ágil y uno tradicional, y constituyen las ideas centrales del desarrollo ágil.

I. Nuestra mayor prioridad es satisfacer al cliente mediante entregas tempranas y continuas de software con valor. Para que una metodología puede ser calificada como ágil debe empezar a entregar software funcionando y útil en pocas semanas. Esto acaba con la incertidumbre, desconfianza, insatisfacción y desmotivación producidas en el cliente debido a las largas esperas para ver resultados concretos. Por lo tanto, la participación del cliente se hace más productiva en la medida en que el software está siendo probado, revisado y aprobado constantemente por quien lo requirió y lo va a usar.

II. Bienvenidos los cambios a los requerimientos, incluso los tardíos. Los procesos ágiles aprovechan los cambios para la ventaja competitiva del cliente. Es ambicioso esperar que el cliente defina de manera definitiva todos sus requerimientos desde el comienzo y peor aún depender de ello para adelantar el proyecto. Los cambios en los requerimientos deben asumirse como parte del proceso de maduración del software, debe entenderse que cuando el cliente describe una necesidad lo hace desde su perspectiva de usuario y que sus conocimientos técnicos lo pueden limitar para hacerse entender completamente. Por lo tanto, las novedades en los requerimientos pueden ser ajenas a la voluntad del cliente. Esta forma de ver los cambios en los requerimientos

induce al equipo de desarrollo a preferir los diseños flexibles, lo cual aumenta la satisfacción del cliente y redunda finalmente en beneficio del equipo de desarrollo dada la comodidad en el diagnóstico y ajustes que se requieren en la etapa de mantenimiento.

III. Liberar frecuentemente software funcionando, desde un par de semanas a un par de meses, con preferencia por los periodos más cortos. El cliente siempre espera ver funcionando el programa, y es eso lo que debe entregársele. Pocas veces resulta conveniente, después de varios meses de trabajo, entregar sólo informes, modelos abstractos y planes. Se deben entregar resultados que incluyan software que el usuario pueda ver trabajando. Si hay una circunstancia que motiva al cliente es poder usar el software que solicitó.

IV. Las personas del negocio y los desarrolladores deben trabajar juntos diariamente a lo largo del proyecto. Si bien el usuario desconoce lo referente al lenguaje, el diseño de bases de datos, protocolos y demás aspectos técnicos, es él, quien nos puede señalar qué está bien desde el punto de vista de la funcionalidad y resultados entregados por el software. La intervención oportuna del usuario puede resultar decisiva en el éxito de un proyecto y puede reducir el costo o el tiempo. Esta intervención puede ser en cualquier momento, por lo cual el usuario debe estar involucrado todo el tiempo que dure el proyecto.

V. Construir proyectos en torno a individuos motivados. Darles el entorno y apoyo que necesiten, y confiar en ellos para que consigan hacer su trabajo. El ánimo, el sentido de pertenencia y la disposición del equipo de trabajo son fundamentales en un proyecto de software. Parte de la motivación está en la confianza que se muestre en el equipo de trabajo, el respeto por sus aportes y la comodidad que se les conceda en el momento de realizar su trabajo. Todo lo que se pueda hacer por dar ánimo y motivación a las personas participantes en el proyecto debe hacerse.

VI. El método más efectivo y eficiente de compartir información a, y dentro de un equipo de desarrollo, es la conversación cara a cara. El trabajo en equipo debe apoyarse con un buen sistema de comunicación tanto entre los miembros del equipo de desarrollo como entre éstos y el usuario. La mejor forma de hacerlo es hablando personalmente; en la medida en que se evitan los intermediarios en el proceso de comunicación, como son el papel, el teléfono, el sistema de correo, y demás medios de comunicación, se incrementa la posibilidad de que el resultado sea el que se solicitó.

VII. El software funcionando es la medida de progreso. Cuando se trata de establecer el estado de un proyecto, si bien existen diversas formas de medirlo, es la cantidad de requerimientos implementados y funcionando la que más claridad y confiabilidad ofrecen para establecer una medida del avance del proyecto. Cualquiera otra que se presente será superada por una que involucre el software qué ya ha sido probado y aprobado por el usuario.

VIII. Los procesos ágiles promueven el desarrollo sostenible. Los patrocinadores, desarrolladores y usuarios deberían ser capaces de mantener relaciones cordiales. Se debe trabajar de forma que lo urgente no se imponga sobre lo importante. Desde el inicio del proyecto se debe asignar responsabilidades y tareas de manera que siempre se puedan cumplir.

IX. La atención continua a la excelencia técnica y al buen diseño incrementan la agilidad. Además de satisfacer los requerimientos del usuario, los aspectos técnicos deben ser excelentes, independientemente de su cantidad y complejidad. La calidad debe ser vista desde dos perspectivas, la del usuario y la del equipo desarrollador. Para el personal técnico resulta evidente que cuanta más calidad tenga el software en cuanto a diseño y estándares de implementación, más rendimiento obtiene en las tareas de pruebas, mantenimiento, y mayor reusabilidad.

X. La simplicidad – el arte de maximizar la cantidad de trabajo no hecho- es esencial. Se estima que el cliente nunca usará el 90% de la funcionalidad que se implementa sin que está haya sido solicitada. Se deben centrar los esfuerzos en lo que realmente importa, de manera simple, sin excederse en refinamientos y optimizaciones innecesarias. Si funciona así, déjelo así, si se va a perfeccionar u optimizar una rutina o programa se debe evaluar minuciosamente el costo beneficio.

XI. Las mejores arquitecturas, requerimientos y diseños emergen de los equipos auto-organizados. Los principios que rijan en equipo de trabajo deben surgir de su interior, los ajustes, estructuras administrativas deben formularse con la participación de todo el equipo teniendo siempre presente el bien colectivo, la responsabilidad es de todos.

XII. En intervalos regulares, el equipo reflexiona sobre cómo volverse más efectivo, entonces afina y ajusta su comportamiento como corresponde. El equipo de trabajo está todo el tiempo dispuesto a cambiar lo que sea necesario para mejorar. En cada tarea siempre existe la posibilidad de hacerlo mejor la próxima vez.

2.5. Características comunes de los métodos ágiles

Las metodologías ágiles impulsan generalmente una gestión de proyectos que promueve el trabajo en equipo, la organización y responsabilidad propia, un grupo de buenas prácticas de ingeniería de software que brindan una entrega rápida de software de alta calidad, y un enfoque de negocios que alinea el desarrollo con las necesidades del cliente y los objetivos de la compañía. Existe una gran variedad de metodologías ágiles cada una tiene sus particularidades, manejan roles muy peculiares, suponen un conjunto de expertos con habilidades para resolver desde problemas técnicos hasta los más abstractos y complejos, por lo tanto exigen experiencia. Definen ciclos, que en algunos casos incluyen varias fases, similares a los procesos de desarrollo de software tradicionales. Generan documentación indispensable, aunque el proceso para generarlas es menos rígido. Aunque lo más importante es que están abiertas a modificar requerimientos, sin importar su impacto en la arquitectura del sistema. Una comparación básica entre metodologías ágiles y metodologías tradicionales puede verse en las siguientes tablas. Estas diferencias no solo afectan al proceso de desarrollo de software, sino también al contexto del equipo y a la organización.

	Métodos Ágiles	Métodos Tradicionales
Enfoque	Adaptación	Predictivo
Éxito de Medición	Valor del Negocio	Conformación de planificar
Tamaño del proyecto	Pequeño	Grande
Estilo de gestión	Descentralizada	Autocrático
Perspectiva para el Cambio	Cambio y Adaptabilidad	Cambio y Sostenibilidad
Cultura	Liderazgo-Colaboración	Comandos de control
Documentación	Bajo	Pesado
Énfasis	Orientada a las personas	Orientado a los procesos
Ciclos	Muchos	Limitado
Dominio	Impredecible exploratorio	Previsible
Planificación por adelantado	Mínimo	Exhaustivo
Retorno de la Inversión	A principios de Proyecto	Fin de Proyecto
Tamaño del equipo	Pequeños / Creatividad	Grande

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Tabla 1. Diferencias entre metodologías ágiles y tradicionales

Las metodologías ágiles son una alternativa interesante para superar las debilidades de las metodologías convencionales, pero, al igual que los computadores no son en sí mismos la solución a los problemas de procesamiento de información, éstas no son la solución a todos los problemas que enfrenta el desarrollo de software. Con el surgimiento de las metodologías ágiles, el concepto de etapa se desvanece dando paso a la idea de actividades, las cuales pueden ser organizadas a comodidad del equipo de trabajo, en paquetes pequeños conservando las mismas labores e identidad de las etapas concebidas en las primeras metodologías. Es claro que la informática y la computación son ciencias nuevas, sin embargo ya es tiempo de que se les concedan espacios diversos y espontáneos en los que las personas que participan en los proyectos del área puedan sentirse más productivos con base en las circunstancias particulares de cada proyecto.

La participación directa del usuario durante todo el proyecto es una garantía de éxito, ya que la detección de los errores es más oportuna y de igual manera su corrección Si bien los proponentes presentan su experiencia como muestra de conveniencia en el uso de sus metodologías, en términos generales éstas ofrecen mayores probabilidades de éxito en proyectos que cumplan ciertas condiciones como son no involucrar a más de 15 o 20 programadores, que no requieran más de cinco o seis meses de trabajo, que el entorno de desarrollo sea apropiado para la comunicación entre los miembros del equipo, que el cliente o usuario esté dispuesto y disponible para el trabajo en equipo con el personal técnico y que los requerimientos puedan ser formulados según va avanzando el proyecto.

Podemos destacar 9 pautas para acortar la duración o ciclo de vida de un proyecto:

1. Modular.
2. Iteratividad con cortos ciclos permitiendo verificaciones y correcciones rápidas.
3. Fijar tiempos (time-bound) con ciclos iterativos de 1 a 6 semanas.
4. Eliminar todas las actividades innecesarias.
5. Adaptarse con posibles nuevos riesgos que puedan aparecer.
6. Utilizar un método incremental para construir la aplicación en pequeños pasos.
7. Un enfoque convergente e incremental minimiza los riesgos.
8. Orientado a las personas, no a los procesos ni a la tecnología.
9. Forma de trabajar comunicativa y cooperativa.

Se propone el desarrollo iterativo como el denominador común de los procesos ágiles. Los nuevos requisitos pueden ser introducidos, modificados o eliminados en las sucesivas iteraciones. Este enfoque que puede retrasar la finalización del proyecto, necesita un contrato que permita estas dilataciones en los plazos de entrega.
Es muy importante tener satisfecho al cliente cuando se le entregue el proyecto y no sólo en el momento del inicio de éste con un contrato. Son necesarios métodos que acepten los cambios del cliente, y para ello, proponen unas reglas:

- Realizar la primera entrega en semanas, para lograr una victoria rápida y obtener el imprescindible feedback del cliente.
- Inventar soluciones simples, de forma que haya menos que cambiar y hacer las cosas más fáciles.

- Mejorar la calidad del diseño continuamente, haciendo la siguiente iteración más fácil de implementar. - Testear constantemente para detectar fallos pronto y ahorrar dinero.

Un planteamiento para procesos ágiles puede ser:

- La gente cuenta.
- Es posible realizar menos documentación.
- La comunicación es un asunto crítico.
- Las herramientas de modelado no son tan útiles como se pensaba normalmente.

Puntos comunes a todos los métodos que se consideren ágiles:

AGILIDAD: "Ágil" significa ser capaz de moverse de forma rápida pero con decisión, reaccionar ante los cambios con velocidad y destreza y cambiar la dirección manteniendo el equilibrio. Este equilibrio contrasta con métodos más tradicionales que se asemejan a una disciplina militar.

CAMBIO Los métodos ágiles tratan los cambios como algo esperado, los aceptan y valoran mucho porque usan su información como feedback para adaptarse y ser más eficientes. Por contra, los métodos tradicionales, ven los cambios como enemigos y dedican mucho esfuerzo en controlarlos para poder retomar el plan inicial.

PLANNING Cuando un proyecto ágil no avanza como estaba previsto, los cambios o desviaciones que ha habido le aportan información para mejorar el plan. Los métodos tradicionales aplican correctivos para no cambiar el plan. La cuestión es, ¿debería moldearse la realidad para encajar en el plan? o ¿Debería reconstruirse el plan para amoldarse a la realidad? La respuesta más sensata es una poco de cada; en el punto medio está la virtud según Aristóteles.

COMUNICACIÓN Todos los MA (Métodos Ágiles) promueven la comunicación entre los participantes (cliente, usuario final, contratista, etc.) y quitan importancia a la documentación que no ayude en la comunicación. Por tanto, son contrarios a la documentación que acabará archivada y olvidada en la mayoría de los casos. Los métodos tradicionales no están en contra de la comunicación, pero la formalizan en demasía según los MA. Evidentemente, llevado al extremo, la ausencia de documentación sería un problema, por eso se debe tener en cuenta que todo aquello de lo que valga la pena hablar, vale la pena que quede constancia por escrito.

APRENDIZAJE Todos los MA se toman un proyecto como una experiencia de la que aprender. En el comienzo del proyecto, ni los clientes ni los programadores tienen un control absoluto de lo que se debe hacer. También aprenden del feedback del cliente. Los métodos tradicionales intentan que todo quede definido desde el principio, y si aparecen cambios, documentan estos problemas y aplican correctivos.

Como resumen podemos decir que un método de desarrollo de software ágil es:

- Incremental: versiones pequeñas de software, con ciclos rápidos.

- Cooperativo: desarrolladores y cliente siempre en contacto constante.
- Directo: el método en si es fácil de aprender y modificar, bien documentado.
- Adaptativo: capaz de tolerar cambios de última hora.

2.6. Manifiesto de proyecto ágil

El Manifiesto de proyecto ágil, Agile Project Manifest, disponible en www.pmdoi.org, dice así:

> *DECLARACIÓN DE INTERDEPENDENCIA*
>
> *Enfoques ágiles y adaptables para unir personas, proyectos y valor. Somos una comunidad de líderes de proyecto con éxito a la hora de entregar resultados. Para lograr estos resultados:*
>
> *• Aumentamos la ganancia en la inversión (increasing return) centrándonos en el flujo continuo de valor.*
>
> *• Entregamos resultados fiables comprometiendo a los clientes en interacciones frecuentes y propiedad compartida.*
>
> *• Esperamos la incertidumbre y la manejamos gracias a las iteraciones, anticipación, y adaptación.*
>
> *• Liberamos creatividad e innovación reconociendo que los individuos son la máxima fuente de valor, y creando un ambiente donde puedan diferenciarse.*
>
> *• Incrementamos el rendimiento a través de la responsabilidad de grupo para una responsabilidad y resultados compartidos, buscando la efectividad del equipo.*
>
> *• Mejoramos efectividad y fiabilidad a través de estrategias, procesos y prácticas específicas adaptadas a la situación.*
>
> *©2005 David Anderson, Sanjiv Augustine, Christopher Avery, Alistair Cockburn, Mike Cohn, Doug DeCarlo, Donna Fitzgerald, Jim Highsmith, Ole Jepsen, Lowell Lindstrom, Todd Little, Kent McDonald, Pollyanna Pixton, Preston Smith y Robert Wysocki.*

El título "Declaración de Interdependencia" tiene significados múltiples. Significa que los miembros de un equipo son parte de un todo interdependiente y no un grupo de individuos aislados. Significa que los equipos del proyecto, sus clientes, y sus participantes también son interdependientes. Los equipos del proyecto que no reconocen esta interdependencia, raras veces tendrán el éxito. Estos valores también forman un juego interdependiente. Mientras cada uno es importante independientemente de los otros, los seis forman un sistema de valores que proporciona una visión moderna de gestionar los proyectos, particularmente los complejos e indecisos. Las seis declaraciones (valor, incertidumbre, clientes, individuos, equipos, y contexto específico de la situación) definen un todo inseparable. Por ejemplo: es difícil entregar valor sin un cliente que valore algo. Es difícil tener equipos viables sin reconocer las contribuciones individuales. Es difícil manejar la incertidumbre sin aplicar las estrategias específicas

circunstanciales. Cada una de las declaraciones de valor tiene una forma distinta; el porqué un punto es importante precede a la descripción del valor. Por tanto, la “ganancia creciente en la inversión” es la razón de que centrarse en el flujo de valor continuo es importante. Las declaraciones de valor dan énfasis a la importancia de entregar resultados fiables (no es lo mismo que repetibles), gestionando la incertidumbre, liberando la creatividad e innovación, incrementando el rendimiento, y mejorando la efectividad.

Cada uno de los medios de las declaraciones expresa lo que ese grupo piensa y son los aspectos más importantes de la gestión de proyecto moderna, y también intentan diferenciar un estilo ágil y adaptable de gestionar proyectos. Por ejemplo, en la última declaración de valor, la frase “estrategias, procesos y prácticas específicas adaptadas a la situación” (situationally specific strategies, processes, and practices) indican que estos asuntos no deben ser demasiado estándares ni estáticos, sino dinámicos para adaptarse a las necesidades del proyecto y a los equipos. Otros estilos de gestión de proyectos son más propensos a la estandarización y a los procesos preceptivos.

3. FILOSOFÍA LEAN

Este apartado pretende servir de guía para entender la filosofía Lean en su concepción original.

Se pretende dar una visión genérica de los conceptos y principios de la metodología Lean, tal y como nació, para aplicarlos en el siguiente apartado en el ámbito de la ingeniería de software.

3.1. Definición

El término Lean es el nombre con el que se da a conocer en occidente al sistema de producción de Toyota.

Su objetivo fundamental es la satisfacción del cliente, mediante la entrega de productos y servicios de calidad que son lo que el cliente necesita, cuando lo necesita en la cantidad requerida al precio correcto y utilizando la cantidad mínima de materiales, equipamiento, espacio, trabajo y tiempo.

Para lograrlo, los fundamentos del enfoque Toyota son la eliminación del sistema de producción de todo aquello que no añade valor al cliente y el mayor aprovechamiento de la experiencia e inteligencia de las personas, a través de la polivalencia y de su participación en la mejora continua.

Así, lo primero que hay que determinar es precisamente cómo lograr esta satisfacción: qué es valor en términos del cliente. En cualquier proceso, añadirá valor toda aquella transformación (física o de la información) del producto, servicio o actividad en algo que quiera el cliente.

Lean es más que un conjunto de herramientas y prácticas. Éstas, emanan de una serie de principios que deben calar en la cultura de la organización antes que cualquier otra cosa. Más allá de la implantación de un conjunto de herramientas, Lean implicará la transformación de la organización, comenzando precisamente por la adopción de sus principios.

Una forma visual de representar en un mismo diagrama los elementos característicos del sistema Lean es “La Casa”. La siguiente figura es un extracto de la casa de Lean expuesta por Jeffery K. Liker (Liker).

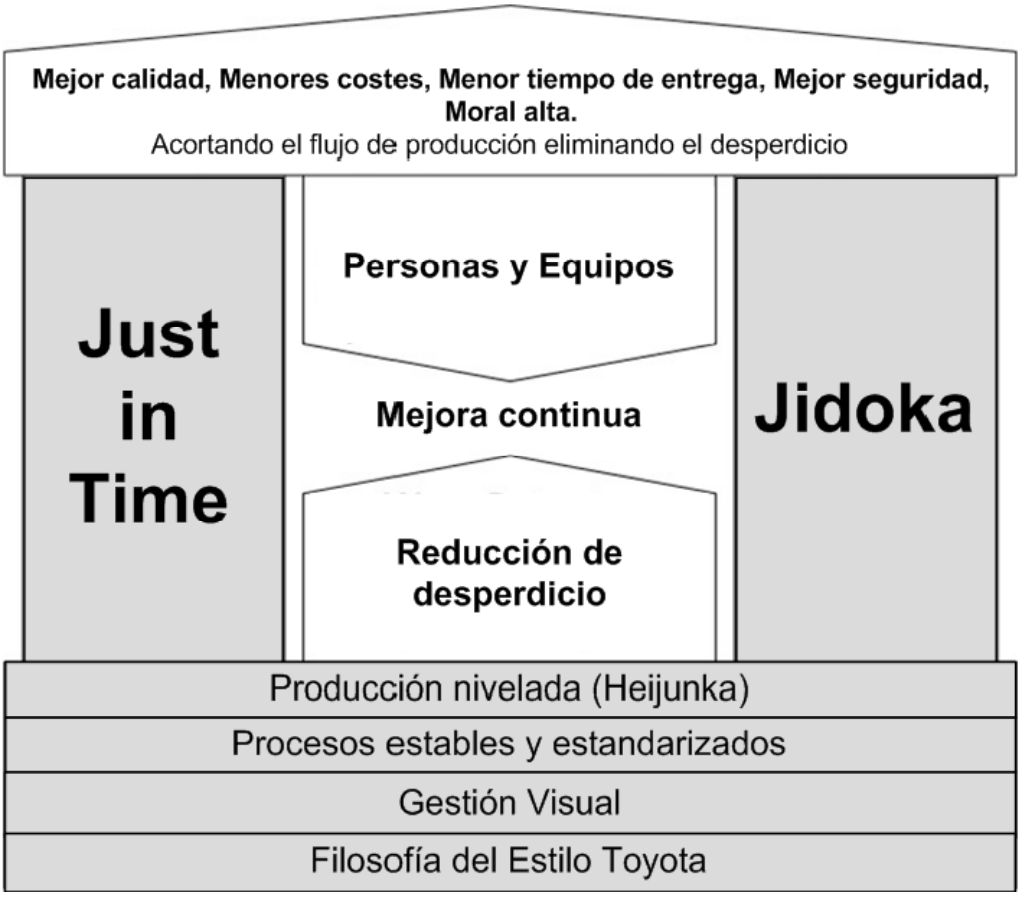

Figura 1 – La Casa de Toyota-Lean (reducida)

Cada elemento representado tiene importancia por sí mismo, pero lo más importante es la forma en que se complementan entre sí.

El techo, representa los objetivos que se persiguen, mejor calidad, menores costes, etc. Los pilares sobre los que se apoya son Just-inTime (producir lo que se necesita y cuando se necesita) y Jidoka (calidad inherente al proceso de producción). En el centro, la mejora continua, reducir el desperdicio gracias a la participación de las personas y equipos. La base sobre la que se sostiene todo el sistema será la filosofía, en primer lugar y conceptos como la gestión visual, la estandarización y el nivelado de la producción.

3.2. Contexto de aplicación

En cuanto a su contexto de aplicación, podemos decir que lean es potencialmente aplicable todas las áreas de una organización, aunque habitualmente se asocia al área de producción. Por otra, respondiendo a qué tipo de productos y servicios ofrecen las organizaciones que adoptan Lean, podemos afirmar que su mayor difusión se da en el ámbito de la manufactura (especialmente en el sector del automóvil), y que se ha

aplicado con éxito otros campos como el de la logística y distribución, construcción, así como van en aumento las referencias de nuevos escenarios, como su aplicación en la sanidad (Lean Healthcare) o en la Administración Pública (Lean Goverment).

3.3. Historia

La filosofía Lean tiene su origen en los inicios del grupo Toyota, y a través de los años ha ido evolucionando en la forma de una serie de preceptos y principios en torno a dos ideas fundamentales:

1. Dar una gran relevancia al papel que ocupa el componente humano de la producción
2. Un manifiesto espíritu de mejora continua.

A continuación se refieren algunos de los hitos que han marcado la historia de esta filosofía.

Los preceptos de la filosofía Toyota (Toyota precepts) fueron establecidos en

1935. en el 5º aniversario de la muerte del fundador del grupo Toyota, Sakichi Toyoda, como compendio de sus enseñanzas y reflejan el espíritu de la compañía.

50s. Durante los años 50, en los años de la reconstrucción tras la segunda guerra mundial, la industria japonesa en general y Toyota en particular tuvieron que enfrentarse a poner en pie la industria de manufactura:

- Con una demanda limitada, sin posibilidad de recurrir a las economías de escala
- En un escenario donde es complicado obtener financiación, y cuestiones como el inventario complican más aún obtenerla.

El escaso mercado y la limitación de los recursos disponibles fueron el caldo de cultivo para un planteamiento más eficiente de producción capaz de dar respuesta a una situación extremadamente adversa.

En lo que a los procesos se refiere, se redefine la producción en base a una serie de ideas fundamentales para dar solución a los problemas expuestos:

Fabricar únicamente lo que se necesita: aquello para lo que hay un cliente. El inventario es dinero inmovilizado ocupando espacio, hay que evitarlo. Fundamento extensible a todas las etapas del proceso: cada paso debe producir exclusivamente lo que necesita el siguiente.

Eliminar aquello que no añada valor al producto: valor entendido en términos del cliente. **Detener la producción si algo va mal**: para localizar la fuente del error inmediatamente y corregirlo para evitar su propagación, pasar del método de inspección a la producción cero-defectos.

Así, los pilares de este sistema serán la producción **Just in Time** (producción de lo que se necesita, cuando se necesita…) y **Jidoka** (calidad inherente al propio sistema de producción), dentro de un clima de mejora continua y declarado respeto a las personas involucradas en el sistema

70s. El éxito de las ideas aplicadas en base a este sistema revitaliza Toyota y se extienden por Japón a partir de los años 50. Su eficacia se da a conocer en occidente durante la década de los 70 durante la crisis del petróleo, el sistema permite la adaptación de la producción para dar respuesta a un nuevo tipo de demanda más rápido y de manera menos traumática que sus competidores, acabando con el dominio que Ford y General Motors habían tenido hasta ese momento en la industria automóvil.

80s. En la década de los 80, Toyota y otras empresas japonesas exportan este sistema de producción a fábricas de Europa y América, comienza a extenderse fuera de Japón y su filosofía comienza a adaptarse más allá de la manufactura.

90s. En 1990 J.P. Womack y D.T. Jones documentan la experiencia Lean en Estados Unidos en su libro "The Machine That Changed the World", exponiendo el impacto de esta filosofía en la industria del automóvil en el contexto económico mundial, en un estudio previo a esta publicación, acuñan el término Lean para referirse al sistema utilizado por Toyota.

1992. Se publican los 7 principios directores (Toyota Guiding Principles), como reflejo del tipo de empresa que Toyota pretende ser: su filosofía de gestión, valores y métodos que ha adoptado desde su fundación.

1996. Unos años más tarde, en 1996, Womack y Jones publican "Lean Thinking" que generaliza las lecciones aprendidas en su publicación anterior describiendo experiencias de implantación de Lean en otros sectores.

1997. Womack funda el Lean Enterprise Institute, organización sin ánimo de lucro cuyo objetivo es la promoción de la filosofía Lean a todos los niveles.

2001. Se crea el Manual de Estilo Toyota "Toyota Way", un documento interno de la compañía donde se resume su filosofía e ideales, y en el que se identifican los dos pilares principales de esta filosofía "Respetar a las personas" y "Mejora continua".

2004. Liker resume el Estilo Toyota en 14 principios que constituyen una hoja de ruta para la aplicación de los valores de la empresa por los todas las personas que forman parte de ella, en su trabajo cotidiano y en sus relaciones con los demás.

Es notorio, que el éxito de esta filosofía durante las últimas dos décadas ha creado una gran demanda de conocimiento acerca de la misma y en consecuencia, se ha dado una pródiga investigación y la publicación de multitud de libros, artículos y todo tipo de recursos relacionados con esta materia. Así mismo, día a día crece su aplicación a nuevos escenarios más allá de la manufactura.

3.4. Principios

Los preceptos de la filosofía Toyota (Toyota precepts) siguen presentes desde el origen de la compañía:

1. Contribuir en conjunto, e independientemente de la posición política, con el desarrollo y bienestar del país a través del cumplimiento cabal de nuestras tareas.
2. Adelantarse a los tiempos mediante una interminable creatividad, curiosidad y perfeccionamiento.
3. Ser práctico y evitar la frivolidad.
4. Ser amable y generoso, crear un ambiente cálido y familiar.
5. Ser respetuoso; mostrar y actuar con gratitud por todas las cosas, grandes y pequeñas.

Los siete principios directores (Toyota Guiding Principles), que reflejan la filosofía de gestión, valores y métodos que ha adoptado Toyota desde su fundación son los siguientes:

1. Honra el espíritu de la ley de todas las naciones para ser un buen ciudadano corporativo del mundo.
2. Respetar la cultura de todas las naciones y contribuir al desarrollo económico y social de todas las comunidades.
3. Dedicarnos a proporcionar productos limpios y seguros y mejorar la calidad de vida a través de todas nuestras actividades.
4. Crear y desarrollar tecnologías avanzadas y proveer de productos y servicios excepcionales que satisfagan las necesidades de nuestros clientes.
5. Promover una cultura corporativa que mejore la capacidad creativa individual y el valor de trabajar en equipo, honrando la confianza y el respeto mutuo entre el trabajador y la dirección.
6. Perseguir el crecimiento en armonía con la comunidad global a través de una gestión innovadora.
7. Trabajar junto a nuestros colaboradores en la investigación y desarrollo para conseguir el crecimiento y beneficio mutuo estable y a largo plazo, manteniéndonos abiertos a nuevas colaboraciones.

Los cinco principios del “pensamiento Lean” identificados por J.P. Womack y D.T. Jones son los siguientes:

1. Definir valor: Desde el punto de vista del cliente, en términos de un producto específico, de características específicas y ofertado a un precio y plazo específico.
2. Identificar la cadena de valor: Eliminar desperdicios, encontrar los pasos necesarios y suficientes para dar el valor al cliente.
3. Crear flujo: Hacer que todo el proceso fluya suave y directamente de un paso que agregue valor a otro, desde la materia prima hasta el cliente.
4. Producir el “tirón” del cliente: Una vez hecho el flujo, producir a la demanda real de los clientes, en lugar de producir según pronósticos.

5. Perseguir la perfección: Una vez que una empresa consigue los primeros cuatro pasos, intentar mejorar continuamente.

Según el Manual de Estilo Toyota “The Toyota Way”, los dos pilares principales de esta filosofía y los cinco términos clave para llevarla a cabo son:

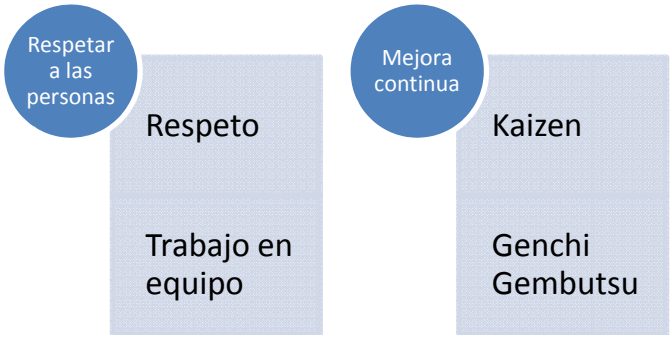

Figura 2 - Los elementos clave del Estilo Toyota

1. Desafíos o retos. La visión a largo plazo y el esfuerzo por afrontar todos los retos con el valor y la creatividad necesarios para hacer realidad esa visión.
2. Kaizen. Esforzarse por “mejorar continuamente. Como ningún proceso puede considerarse nunca perfecto, siempre queda espacio para mejorar”.
3. Genchi Genbutsu. Implica "ir al origen para descubrir los hechos que nos ayuden a tomar decisiones correctas, crear consenso y alcanzar los objetivos marcados".
4. Respeto. Respetar a las personas, el esfuerzo por que la comprensión rija las relaciones dentro de la empresa y con el exterior, aceptando sus responsabilidades y haciendo todo lo posible para crear confianza mutua a partir de una comunicación honesta.
5. Trabajo en equipo.- Estimular el crecimiento personal y profesional, ofrecer oportunidades para el desarrollo y maximizar el rendimiento individual y de los equipos.

Los catorce principios del estilo Toyota identificados por Liker, se agrupan en el llamado Modelo 4P: Problemas, Personas, Procesos y Philosophy (filosofía).

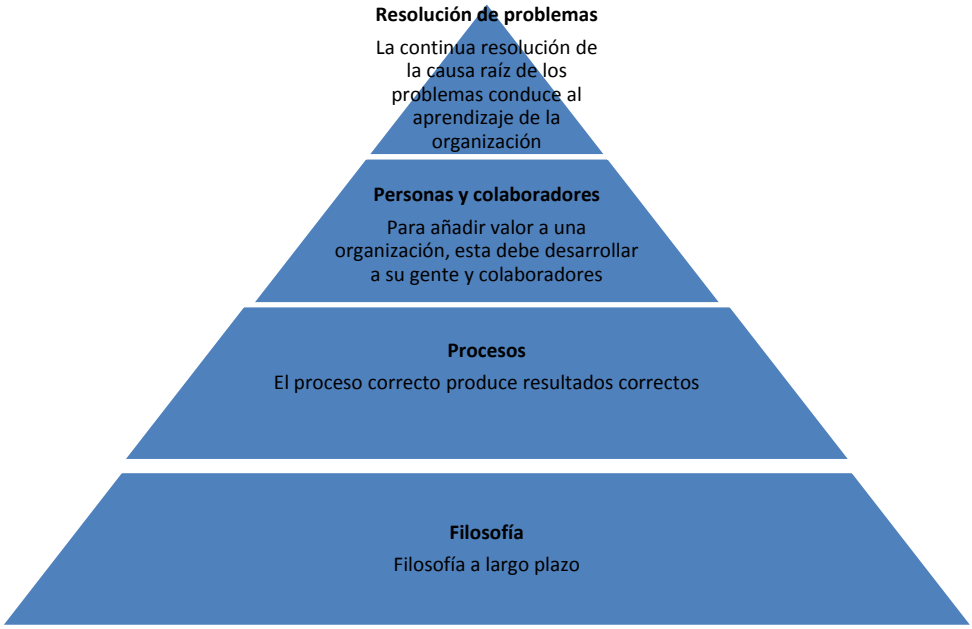

Figura 3 – Principios Toyota, Modelo 4P

Filosofía

1. Basar las decisiones de administración en una filosofía de largo plazo, aún a costo de las metas financieras de corto plazo. Procesos
2. Crear flujos de procesos continuos para llevar los problemas a la superficie.
3. Usar sistemas "Pull" para evitar la sobreproducción.
4. Nivelar la carga de trabajo.
5. Construir una cultura orientada a la solución de problemas, para obtener calidad a la primera vez.
6. La estandarización de tareas y procesos es la base de la mejora continua y la toma de poder por los empleados.
7. El control visual impide que se oculten los problemas.
8. Utilizar tecnología fiable y testeada que sea de utilidad para las personas y los procesos. Personas y Colaboradores
9. Desarrollar líderes que entiendan su trabajo en Toyota, vivan su filosofía y se la enseñen al resto.
10. Desarrollar personas y equipos excepcionales que sigan la filosofía de la compañía.
11. Se debe respetar la red de colaboradores y proveedores, dándoles nuevos retos y ayudándolos a mejorar. Resolución de Problemas
12. Para comprender una situación se debe verificar en primera persona.
13. Tomar decisiones lentas por consenso, considerar profundamente todas las opciones e implementar las decisiones rápidamente.
14. Por medio de la reflexión implacable (hansei) y la mejora continua (kaizen) la empresa debe asumir un rol de aprendizaje sistemático.

3.5. Personas y equipos de trabajo

La verdadera ventaja de esta filosofía reside precisamente en las personas, en la capacidad para aprovechar la inteligencia de los empleados.

Lean se centra en lograr el compromiso y el desarrollo personal y de todas las personas de una organización, en especial aquellos que están más cerca del producto.

El principio que debe catalizar la transformación hacia Lean es precisamente el respeto a las personas.

3.6. Mejora continua – Kaizen

La palabra japonesa para Kaizen se traduce como “cambio para mejor” o “mejora”.

Como se ha expuesto con anterioridad, el espíritu de mejora es uno de los pilares sobre los que se apoya el éxito de esta filosofía.

Kaizen se enfoca en las personas y en la estandarización de los procesos. Su objetivo es incrementar la productividad mediante el control de procesos (reducción de tiempos de ciclo, la estandarización, etc.) y la eliminación de mudas. Fomenta la participación y contribución de las personas que realizan el trabajo en la mejora del sistema (algo que impacta positivamente en la autoestima y motivación del personal) y mejora la actitud y aptitud de directivos y personal para una rápida adaptación al cambio.

Eventos Kaizen

Los eventos Kaizen son una de las técnicas de mejora continua en el ámbito Lean.

Durante un periodo de tiempo (típicamente una semana) un conjunto de personas analizan un determinado proceso a mejorar en su estado actual, desarrollan una visión Lean mejorada y comienzan su implementación.

Entre los participantes del evento se debe incluir al responsable (dueño del proceso) que será el líder y al resto de personas que realizan el trabajo en el mismo. También es aconsejable incluir a representantes de los clientes y proveedores del proceso a mejorar (que podrán ser otros procesos), aunque siempre tratando de involucrar un máximo de 15 personas para que se faciliten los debates e implementación de la mejora.

El evento, se compone de tres fases: la preparación, el propio evento y el mantenimiento y la mejora continua tras el mismo.

Fase 1 – Preparación
La fase de preparación deberá servir fundamentalmente para agilizar la realización del evento kaizen, para esta cuestión, deberán resolverse las siguientes cuestiones:

- Determinar claramente el alcance del proceso, dónde comienza y dónde acaba.
- Establecer los objetivos a alcanzar (mejorar la calidad, reducir costes, etc.) y que estos sean ambiciosos.

- Crear un mapa preliminar del estado actual y colocar una copia del mismo en un lugar visible al equipo de kaizen para facilitar posibles modificaciones y anotaciones al mismo durante el evento.
- Recolectar los documentos relevantes del proceso a mejorar (guías de procedimientos, formularios, etc.)

Fase 2 – el evento kaizen
Comenzará revisándose la información obtenida en la preparación así como con la impartición de cierta formación sobre conceptos Lean si el equipo no está familiarizado con esta filosofía. A continuación, se realiza el evento kaizen:

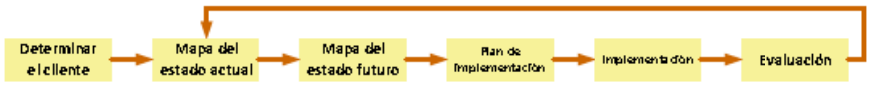

Figura 4 - Actividades de un evento kaizen

Fase 3 – mantenimiento y mejora continua
Una vez realizado el evento, ha de mantenerse la mejora obtenida.
Semanalmente, el equipo se reunirá para asegurarse de la consolidación de la mejora y continuar con la mejora del proceso.
Mensualmente la gerencia revisará el estado de la mejora y dará el reconocimiento del éxito al equipo a medida que se cumplan los objetivos de la implementación.

3.7. Eliminar el "desperdicio"

Desperdicio o "muda" en su concepto más amplio es: "*cualquier otra cosa distinta a la cantidad mínima de equipos, materiales, partes, espacio y tiempo del trabajador que son absolutamente necesarios para dar valor al producto*" - Shoichiro Toyoda

Lean distingue otros tipos de desperdicio: mura y muri.

El Manual de Estilo Toyota, se refiere a la eliminación de las 3M: muda, mura y muri.

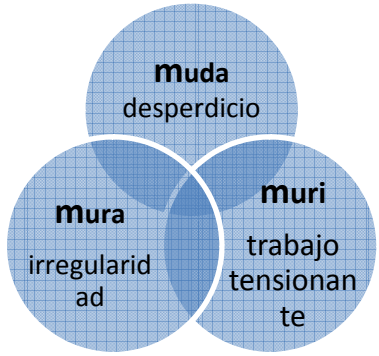

Figura 5 – Las 3M

Mura o irregularidad

Cada vez que se interrumpe el flujo normal del trabajo en la tarea de un operador, el flujo de partes y máquinas o el programa de producción, se dice que existe mura.
El mura está muy relacionado con los cuellos de botella, razón por la que eliminar estas lleva a una mayor fluidez y productividad en los procesos.

Muri o trabajo tensionante

Implica condiciones estresantes para los trabajadores y máquinas, lo mismo que para los procesos de trabajo. Si a un trabajador recientemente contratado se le asigna la tarea de un trabajador veterano, sin dársele antes el entrenamiento suficiente, el trabajo será estresante para él, y es posible que esta persona sea más lenta en sus labores, e incluso puede cometer mayor número de errores.

Tanto el mura como el muri dan lugar a mayor nivel de muda, producto de las irregularidades y tensiones existentes. Identificarlas y contribuir a su disminución y/o eliminación permitirá importantes ahorros de recursos al bajar los niveles de muda. muda desperdicio muri trabajo tensionante mura irregularidad

3.7.1. Tipos de muda

Los siete tipos de muda a los que se hace referencia clásicamente son los siguientes:

Exceso de inventario
Se refiere al almacenamiento excesivo de materia prima, producto en proceso o producto terminado, causando mayores plazos de entrega, costes de almacenamiento, etc.

Sobre-procesamiento o procesamiento incorrecto
Supondrá toda aquella actividad innecesaria o incorrecta que se realice sobre el producto (que no le aporte valor o provoque un defecto en el mismo).

Sobreproducción
Es la producción de bienes o servicio más allá de la demanda de los clientes, aumentando los costes de almacenamiento, transporte innecesario, etc.

Transporte innecesario
Es el movimiento innecesario de productos o materiales entre operaciones.

Esperas
Son aquellos retrasos y tiempos muertos en los que no se está dando valor al producto.

Movimientos innecesarios
Movimientos físicos innecesarios que el personal realiza durante su trabajo: buscar, desplazarse, etc.

Defectos
Se refiere al coste de reacondicionar partes en proceso o productos ya terminados, y el reciclaje o destrucción de productos que no reúnen las condiciones óptimas de calidad.

Liker identifica una octava muda:

No utilizar la creatividad de los empleados

Perder el tiempo, destrezas, ideas, mejoras y oportunidades de aprendizaje por no fomentar la participación o escuchar a los empleados.

Otros conceptos relacionados con la muda en el ámbito de Lean, son los siguientes:

Genchi gembutsu
Este concepto genchi (lugar) genbutsu (producto) se interpreta como "ir al lugar y observar la situación para entenderla"

Ojos para el desperdicio
Se refiere a que se debe estimular en la organización que todos sus miembros participen en la detección de desperdicios en el trabajo diario y en su eliminación.

3.8. Heijunka: Producción nivelada

El término heijunka se refiere a la nivelación de la producción, tanto por volumen como el mix de productos.

En lugar de fabricar acorde al flujo de pedidos de los clientes, algo potencialmente variable, se considera el volumen total de pedidos en un periodo y se equilibra la producción, de forma que se fabrique cada día la misma cantidad de cada tipo de producto para cubrir la demanda global.

Figura 6 - Producción sin nivelar frente a producción nivelada

Con esta técnica, se obtienen cuatro beneficios fundamentalmente:

- Flexibilidad, para servir lo que el cliente quiere y cuando lo quiere. Se reduce el inventario.

- Reduce el riesgo de producir productos y que éstos no se vendan. Se produce bajo pedido.

- Equilibrio: en el uso de maquinaria y trabajo de las personas. Si el trabajo se estandariza teniendo en cuenta que unos productos requieren un mayor esfuerzo que otros, la secuencia de actividades puede contemplar que no se encadenen grandes esfuerzos, haciendo el trabajo más llevadero.

- Se equilibra la demanda a proveedores. Esta estabilidad en la producción se transmite a lo largo de la cadena de suministros cuando se utiliza el sistema Just in Time, de forma que los pedidos a proveedores serán siempre similares lo que facilitará que se aprovechen reduciendo sus propios inventarios conocida la demanda habitual.

3.9. Procesos estables y estandarizados

La estandarización de tareas y procesos es uno de los fundamentos de la mejora continua y del empowerment de los empleados.

Taiichi Ohno apunta a que la producción eficiente se sostiene mediante la prevención de la aparición recurrente de defectos, errores de operación y accidentes y por la incorporación de las ideas de los trabajadores. La estandarización es debe consolidar este conocimiento sobre el buen hacer y servir de base estable para la mejora.

Las personas de la organización son formadas para el uso de los estándares que deben seguir en sus puestos de trabajo. Los líderes de cada área deben conocer los estándares aplicables y supervisar que son seguidos por el personal. De esta forma, la aparición de un error en la cadena de producción siguiendo el estándar, motivará la revisión del mismo para evitar que éste vuelva a sucederse.

La estandarización en Toyota es algo más amplio que un conjunto de instrucciones a seguir para realizar una tarea. El consejero y expresidente de Toyota Fujio Cho describe el trabajo estandarizado con 3 elementos: **takt time** (tiempo necesario para realizar el trabajo y cubrir la demanda), **la secuencia de pasos a seguir**, y el **inventario** necesario para que un operario realice su trabajo.

La base para la flexibilidad e innovación en este sistema reside en el reconocimiento del trabajador más allá del operario que recibe órdenes como un potencial analista y solucionador de problemas. Liker hace mención al término “burocracia habilitadora” (por contraste a la “coercitiva”) para referirse al sistema de organización que busca el **empowerment** de los empleados, donde las reglas y procedimientos son elementos habilitadores y la jerarquía es el sustento del aprendizaje organizacional.

En este contexto, los estándares sirven de ayuda a los trabajadores para realizar su trabajo mejor en lugar de elementos de control que suelen ser percibidos con negatividad. La necesidad de participación en el proceso de estandarización se refuerza por la frecuente aversión a la estandarización del trabajo por parte de los profesionales y la idea de que seguir estrictamente los estándares haría que realizaran peor su trabajo, en especial en las actividades creativas y las intelectuales.

De esta forma, el punto crítico en la implementación de la estandarización en una organización es encontrar el equilibrio entre la rigidez del conjunto de procedimientos y reglas a seguir y los elementos que habilitan su participación en la mejora del proceso de producción. La clave se encuentra en cómo se definen estos estándares y quiénes contribuyen a esta definición, de forma que: Deben ser suficientemente específicos para ser guías útiles, pero con un cierto grado de flexibilidad. La gente que realiza el trabajo debe tener voz para mejorar el estándar.

La incorporación de una mejora propuesta al estándar es un elemento extremadamente motivador.

3.10. Gestión visual

La gestión visual complementa a las personas, dada nuestra naturaleza sensitiva. Así, los mejores indicadores serán aquellos que en lugar de trabajo nos informen claramente el cumplimiento o incumplimiento de la normalidad mediante el uso de nuestros sentidos (sonidos, imágenes, etc.)

El uso de controles visuales eficaces mejora la productividad, reduce los defectos y equivocaciones, facilita la comunicación, dando a las personas mayor control sobre el entorno en el que desarrollan su labor.

Así, la filosofía lean aboga por el uso de controles visuales para impedir que los problemas pasen desapercibidos.

3.11. Just in time

Se entiende por Just in Time (JIT) el conjunto de principios y técnicas que permiten a una empresa la producción y entrega de productos en pequeñas cantidades, con plazos de entrega reducidos, y para dar respuesta a necesidades específicas de los clientes, esto es, entregar el producto correcto, en la cantidad correcta y en el plazo correcto. A continuación se exponen algunos de los conceptos relacionados.

3.11.1. Takt time planning

Takt-time se define como la cadencia a la cual un producto debe ser fabricado para satisfacer la demanda del cliente dada la capacidad productiva. Es importante que el ritmo de producción se ajuste al tack time para evitar el desperdicio: si es más rápido aumenta el inventario y si es más lento se necesitará acelerar la producción, realizar horas extra o disponer de un mayor inventario para cubrir la demanda.

3.11.2. Sistema Pull

Un proceso PULL es aquel que produce según se requiere por el siguiente proceso, es decir, en función de su demanda real, siendo la primera demanda de todo el sistema, la impuesta por el cliente.

En una línea de producción, es interesante que los procesos que la conforman se comuniquen entre sí para que el sistema al completo opere en modo pull, de forma que se minimicen los riesgos de sobreproducción o el desabastecimiento de algún proceso.

3.11.3. Quick Changeover - SMED

Se conoce como “Changeover” o cambio de utillaje en un dispositivo de producción al conjunto de operaciones que se desarrollan desde que se detiene la máquina para proceder al cambio de lote hasta que la máquina empieza a fabricar la primera unidad del siguiente producto en las condiciones especificadas de tiempo y calidad.

En cuanto a Quick Changeover o cambio rápido, es aquel conjunto de técnicas para reducir el tiempo requerido para realizar estos cambios. Este concepto también se conoce como SMED (Single Minute Exchange of Die), donde el cambio debe realizarse en menos de 10 minutos aunque en la actualidad el objetivo de reducción es mucho más ambicioso: una máquina dispone de más de un ajuste predeterminado seleccionable mediante algún tipo de interruptor, o One Touch Setup (OTS) o One Touch Exchange of Dies (OTED).

Son objetivos del cambio rápido:

- Habilitar la producción de lotes más pequeños sin aumento costes Reducir inventario Mejorar la calidad del producto
- Reducir desperdicios (tiempo, movimientos y material)
- Incrementar la flexibilidad de la planta
- Mejorar en el tiempo de entrega del producto

La aplicación de SMED consta de tres etapas:

1. Separar la preparación online de la offline.
2. Convertir en operaciones online tantas operaciones offline como sea posible.
3. Optimizar el conjunto de operaciones offline minimizando el tiempo para realizarlas

3.11.4. Kanban

Idealmente, el flujo de una pieza sería un sistema que, sin la necesidad de inventario alguno, el valor se entrega al cliente en el momento que lo necesita.

A veces, esto no es posible (por ejemplo, por la variabilidad en el tiempo de proceso) y a menudo la mejor solución es utilizar Kanban.

Kanban es un sistema visual de control de la producción que limita la cantidad de trabajo en curso (Work In Progress – WIP).

Funciona de forma similar a una orden de trabajo (nos da información acerca de que se va a producir, en qué cantidad, etc.).

Mediante señales visuales, la demanda se va propagando por los procesos de la cadena de valor, cada proceso conoce en el momento preciso la cantidad exacta que debe producir, evitando inventarios innecesarios. El trabajo fluye arrastrado por la demanda, orquestado por la sucesión de señales provista por Kanban.

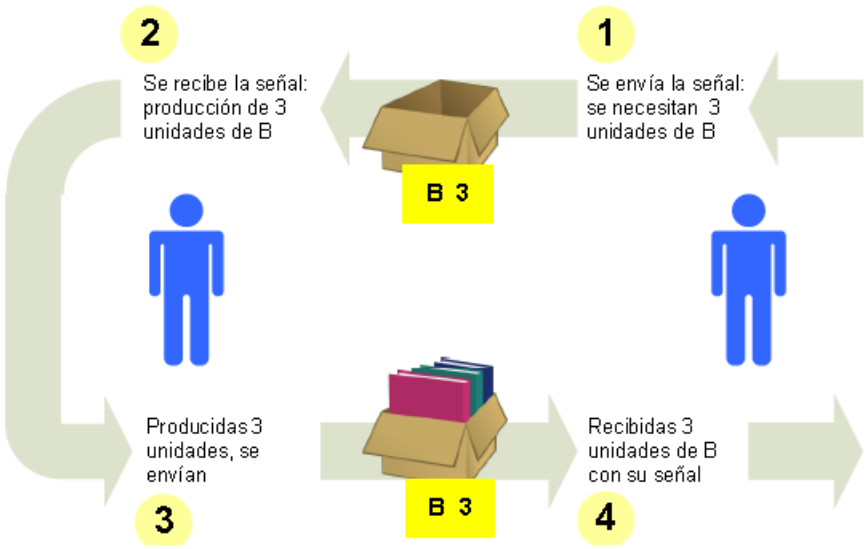

Figura 7 - Sistema Kanban

3.11.5. VSM

El mapa de la cadena de valor representa en un diagrama el conjunto de actividades y procesos y el flujo de materiales e información que rodean el proceso transformación de un producto o prestación de un servicio desde que se recibe la petición del mismo hasta que se realiza su entrega, pudiendo incluir actividades de clientes y proveedores que intervengan en el proceso de producción.

Típicamente, para realizar mapeo del flujo de valor, se siguen una serie de pasos:

1. Seleccionar una familia de productos, entendida como un conjunto de productos que se producen de forma similar, tanto por los medios utilizados como el propio proceso.
2. Formar el equipo que participará en el análisis.
3. Representar los procesos de producción que se siguen para producir el producto, identificando una serie de valores clave para cada uno de ellos: tiempo de ciclo, número de operarios involucrados, etc.
4. Representar el flujo de material, cómo se mueve el material de un proceso a otro, identificando, si existen, los inventarios que se utilizan y su volumen, así como el flujo de materia prima que llega desde los proveedores y de la entrega del producto al cliente.
5. Representar el flujo de información entre los distintos actores involucrados, empresa (u otras unidades organizativas dentro de la misma si es necesario distinguirlas), proveedores, clientes, etc.
6. Calcular los Lead Time, del producto y del proceso. El mapa realizado permitirá visualizar la situación global del sistema de producción y ayudará a reconocer focos de desperdicio (sobreproducción, tiempos de espera, inventarios, etc.).

Conocidos estos, se debe realizar un mapa de cadena de valor a futuro, con un enfoque Lean, ajustando la producción a la demanda de manera eficiente.

3.12. Jidoka

El segundo pilar de la filosofía Lean, parte del principio de que la calidad debe ser inherente al propio sistema de producción.

En este sentido, su carácter debe ser preventivo y no ha de limitarse a la verificación post proceso: cuando se detecta un defecto, se detiene el proceso de producción, se localiza y corrige la fuente del error, para evitar que éste vuelva a producirse. La calidad se incorpora al sistema como mejora del proceso de fabricación.

Taiichi Ohno define Jidoka como “automatización con un toque humano”.

En esencia, se compone de dos partes:

- Un mecanismo de detección de problemas (anormalidades o defectos)
- Un mecanismo para interrumpir el proceso cuando se detecta uno de estos problemas.

Esta detención podrá realizarse:

- Manualmente: por el trabajador al detectar el error
- Mediante paradas automáticas: la automatización de la detección y detención, en caso de error, de una máquina o proceso, sin necesidad de intervención humana.

El “toque humano” tendrá como finalidad hacer que todo vuelva a la normalidad, identificando y corrigiendo la causa raíz del error y que con ello el proceso pueda continuar con calidad.

3.12.1. A prueba de errores (Poka-yoke)

Son mecanismos de calidad preventiva, desarrollados para evitar los errores humanos que deriven en condiciones inadecuadas de operación y, por tanto, fuentes potenciales de error.
Son ejemplos de errores humanos que pretenden evitar: Errores por olvidos, desconocimiento o inexperiencia, de identificación, voluntario, por despiste, por lentitud, falta de estándares, por sorpresa o intencionales.

Pueden eliminar los efectos en dos posibles estados:

- Antes de que ocurran (PREDICCIÓN): se trata de diseñar mecanismos que avisen al operario cuándo se va a cometer un error para que lo evite (ALARMA), que paren la cadena cuando se ha hecho algo mal (PARADA) o que simplemente incorporen nuevos elementos al puesto de trabajo que hagan imposible o difícil un determinado error (CONTROL).

- Una vez ocurridos (DETECCIÓN): se trata de diseñar mecanismos que avisen cuando se ha fabricado un producto defectuoso (ALARMA), que paren la cadena si esto ocurre (PARADA) o que simplemente eviten que ese producto defectuoso pase al siguiente proceso (CONTROL). Muchas de estas técnicas hacen posible la inspección al 100% incorporando mecanismos económicos.

Características de un buen Poka-yoke son:

- Ser simple y económico
- Ser parte del proceso (inspección al 100%)
- Estar próximo al problema para el que se diseñan, facilitando una rápida detección del error por el operario y su actuación para solucionar el problema.

Otras consideraciones acerca de los Poka-yokes:

- Su implantación puede llevar consigo cambios en el proceso.
- Se diseñan para un problema en particular.
- Avisan del problema, siendo del trabajador la responsabilidad de corregirlo.
- Abordan aspectos necesarios pero no suficientes para garantizar la ausencia de errores

3.12.2. Resolución de las causas raíces de los problemas

Para conseguir que la calidad sea inherente al proceso, cuando se detecta un problema es fundamental localizar la causa raíz que lo produce y tomar las acciones oportunas para eliminarla.

Así, Toyota utiliza un proceso en siete pasos para la resolución de problemas al que denomina “Resolución práctica de problemas”, representado en la siguiente figura:

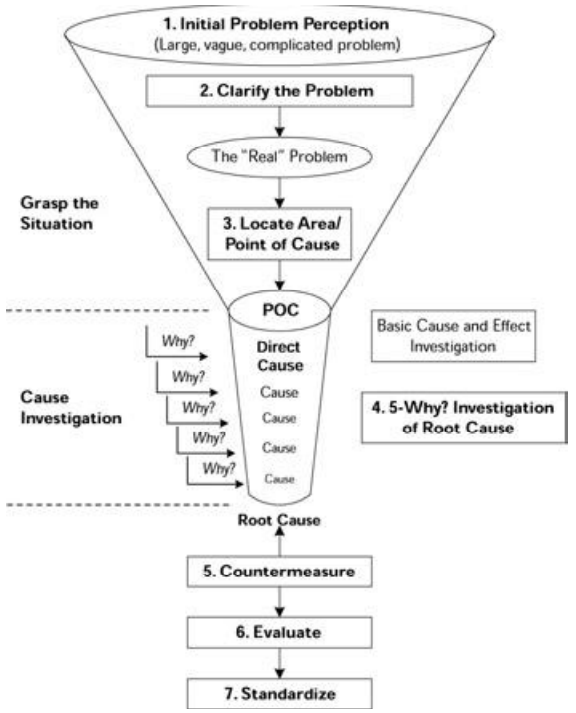

Figura 8 - Resolución práctica de problemas

El ciclo de Deming y el uso de la técnica 5 Why's son característicos de esta técnica. Definido y localizado dónde ocurre el problema, se investiga su causa original y su posible solución, se aplica, se mide el resultado y si cumple su cometido, se estandariza y se incorpora al proceso de producción.

5 Why's

Los 5 "porqués" es una técnica de análisis de causas de un problema. Consiste en preguntar el por qué de un problema de una forma recursiva 5 veces (este número es solo una forma de forzarnos a profundizar en la explicación)

Su uso combinado con el diagrama de Ishikawa (diagrama causa efecto), permite además organizar y relacionar las causas de un problema.

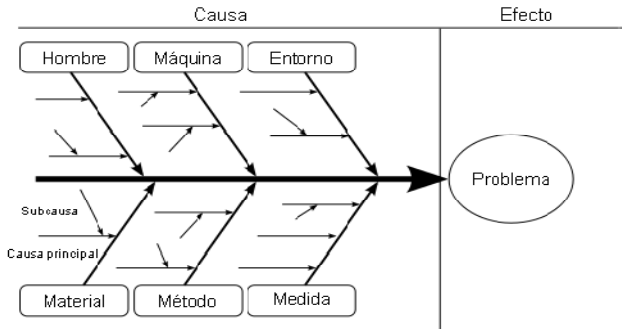

Figura 9 - Diagrama causa-efecto

3.12.3. 5S

Es una filosofía de trabajo diario para lograr un óptimo lugar de trabajo (gemba), que permita la producción eficiente y efectiva.

El término 5s se deriva de cinco palabras japonesas que conforman los pasos a desarrollar para obtener este resultado:

- Seiri: diferenciar entre los elementos necesarios de aquellos que no lo son. Implica separar lo necesario de lo innecesario y eliminar o erradicar del gemba esto último.
- Seiton: disponer de manera ordenada todos los elementos que quedan después del seiri. El seiton lleva a clasificar los ítems por uso y disponerlos como corresponde para minimizar el tiempo de búsqueda y el esfuerzo. Para hacer esto, cada ítem debe tener una ubicación, un nombre y un volumen designados. Debe especificarse no sólo la ubicación, sino también el número máximo de ítems que se permite en el gemba.
- Seiso: significa limpiar el entorno de trabajo, incluidas máquinas y herramientas, lo mismo que pisos, paredes y otras áreas del lugar de trabajo. También significa verificar. Un operador que limpia una máquina puede descubrir muchos defectos de funcionamiento. Cuando la máquina está cubierta de aceite, hollín y polvo, es difícil identificar cualquier problema que se pueda estar formando. Sin embargo, mientras se limpia la máquina podemos detectar con facilidad una fuga de aceite, una grieta que se está formando en la cubierta, o tuercas y tornillos flojos. Una vez reconocidos estos problemas, pueden solucionarse con facilidad. Se dice que la mayor parte de las averías en las máquinas comienzan con vibraciones (debido a tuercas y tornillos flojos), con la introducción de partículas extrañas como polvo, o con una lubricación o engrase inadecuados.
- Seiketsu: significa mantener la limpieza de la persona por medio de uso de ropa de trabajo adecuada, lentes, guantes y zapatos de seguridad, así como mantener un entorno de trabajo saludable y limpio. También implica continuar trabajando en seiri, seiton y seiso en forma continua y todos los días.
- Shitsuke: construir autodisciplina y formar el hábito de comprometerse en las 5S mediante el establecimiento de estándares.

4. LEAN APLICADO A LA INGENIERÍA DEL SOFTWARE

La literatura revisada acerca de la aplicación de la filosofía Lean a la Ingeniería del Software pone de manifiesto que estamos hablando de una disciplina emergente cuyo interés va en aumento en los últimos años.

Una de las carencias actuales de la investigación y publicaciones de la aplicación de Lean, quizá la más relevante es la escasa investigación existente que pueda servir de guía a las organizaciones para adoptar estas nuevas prácticas en la Ingeniería del Software, la falta de estudios que cubran el proceso completo (incluyendo tanto parte técnica como la de gestión) y el enfoque que más se aproxima es aquel que fundamentalmente trata la reducción del desperdicio y mejora de la calidad. En este sentido, sólo se han encontrado dos artículos que tratan la implementación con éxito de la filosofía Lean en organizaciones reales: en una compañía de la lista Fortune 500, Capital One, y en Fujitsu Software Technologies.

La primera referencia obligada cuando hablamos de Lean en este ámbito es el trabajo de los hermanos Tom y Mary Poppendieck, a los que se podría atribuir el acuñado del término Lean Software Development. La publicación en 1993 de “Lean Software Development: An Agile Toolkit” y su avance en publicaciones posteriores, como el más reciente “Leading Lean Software Development: Results are Not the Point”, son referenciados en gran parte de la literatura que existe acerca de este tema.

Si bien ya los hermanos Poppendieck exponen cómo se complementan conceptos de la filosofía Lean con las metodologías ágiles como XP o Scrum, los principios del Manifiesto Ágil son el segundo factor común de prácticamente toda la literatura revisada, de manera más o menos directa. R. Morien afirma que las aproximaciones ágiles a la gestión de proyectos parecen tener raíces en el pensamiento y prácticas de la manufactura Lean.

Mientras el pensamiento ágil realiza su enfoque en el desarrollo de software y la gestión de dicho desarrollo, Lean es aplicable a todos los ámbitos, desde el propio desarrollo hasta la misma empresa donde se produce, pasando por clientes y proveedores. En este sentido, las metodologías ágiles desde la perspectiva Lean, son prácticas válidas.

Entre las ventajas que puede ofrecer un paradigma híbrido Ágil + Lean es de resaltar que incorporar la filosofía Lean a las metodologías Ágiles no requiere un gran esfuerzo y se mejoran los resultados a medio – largo plazo, gracias fundamentalmente al enfoque hacia la perfección que incorpora Lean. Otra de las mezclas estudiadas es la aproximación Scrum, CMMI de nivel 5 y prácticas Lean para la mejora continua.

Entre los artículos y libros revisados son más frecuentes aquellos que se enfocan de manera específica en alguna de las técnicas o principios de Lean a la Ingeniería de Software: Jidoka, Kanban, Muda, etc. y aquellos de carácter general, que pretenden dar una visión global de la aplicación de Lean a esta disciplina en alguna de sus variantes.

Entre estos últimos cabe destacar un estudio sobre la aplicación de Lean en el ámbito del Software as a Service (SaaS), uno de los artículos más novedosos de cuantos se ha revisado, dada la creciente demanda de este tipo de soluciones entre los consumidores del sector TIC.

Para abordar este apartado, se expondrán a continuación la extrapolación de elementos de la filosofía Lean al ámbito de esta disciplina. Para facilitar la lectura, se han agrupado en los siguientes subapartados, si bien algunos de los conceptos tratados podrían considerarse en más de uno de ellos:

- Principios
- Personas y equipos de trabajo
- Mejora continua
- Eliminar el desperdicio
- Heijunka – producción nivelada
- Just-in-Time
- Jidoka
- Experiencias de implementación

4.1. Principios

La primera aproximación a la filosofía Lean sobre la que se tiene referencia, si bien ésta no es explícita en su concepción, es el manifiesto ágil. En Febrero de 2001 se reunieron 17 expertos de la industria del software acuñan el término "Ágil" para hacer referencia a nuevos enfoques metodológicos. En esta reunión se crea "The Agile Alliance", organización para promover este nuevo enfoque y se redacta lo que se conoce como "Manifiesto ágil", que plasmará sus valores y principios:

Valores

- Individuos y relaciones sobre procesos y herramientas
- Software funcionando sobre documentación excesiva
- Colaboración con el cliente sobre la negociación contractual
- Respuesta al cambio sobre seguir un plan

Principios

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.

9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

Los 7 principios del desarrollo Lean de Software (Lean Software Development) enunciados por Poppendieck en 2003 son los siguientes:

1. **Eliminar el desperdicio**: hacer desaparecer del proceso y el producto todo aquello que no aporta valor al cliente.
2. **Amplificar el aprendizaje**: ha de fomentarse y facilitarse, reconociendo el autor la naturaleza predominantemente intelectual de la producción de software y la importancia del aprendizaje para mejorar los resultados.
3. **Decidir tan tarde como sea posible**: dada la frecuente incertidumbre que rodea la toma de requisitos, lo más aconsejable es retrasar las decisiones tratando de tomarlas con la mayor cantidad de información posible, y siempre adoptando una aptitud previsora ante la certeza del cambio.
4. **Entregar tan rápido como sea posible**: consecuencia de lo anterior, es necesario disponer de medios que permitan, una vez tomada una decisión, materializarla, sin sacrificar la calidad.
5. **Delegar la responsabilidad al equipo**: el *empowerment*, dotar a aquellos en los que reside el conocimiento y realizan el trabajo del liderazgo suficiente para tomar decisiones y realizarlo, evitando pasos adicionales de aceptación a otras instancias que obstaculizan el flujo normal de actividad.
6. **Construir con integridad**: integridad conceptual, dado que debe responder a una necesidad del cliente, éste debe percibir el producto como algo coherente, donde los requisitos a los que da solución se observan como un todo cohesionado armónicamente. Además, se necesita integridad técnica: una arquitectura coherente, usable, que responde a su cometido y a la que se puede dar mantenimiento, adaptar y ampliar.
7. **Visión global**: se debe evitar la tendencia a realizar mejoras locales a favor de un enfoque global.

En 2006, Poppendieck revisa estos principios y redefine alguno de ellos:

1. **Eliminar el desperdicio**
2. **Calidad integrada:** el desarrollo ha de realizarse desde el primer momento con calidad. Las acciones correctivas han de emprenderse lo más próximo a que se detecta su necesidad y lo que es más importante, debe existir un enfoque preventivo: se deben buscar las condiciones que eviten si quiera la posibilidad de que se den errores.
3. **Crear conocimiento:** el desarrollo de Software es un proceso de creación de conocimiento que va evolucionando a medida que se va produciendo y que, por tanto, se ha de evitar el derroche de tratar de capturarlo prematuramente. Se han de centrar esfuerzos en mejorar este conocimiento, en hacerlo más profundo y en dar respuesta al cambio.
4. **Aplazar las decisiones (decidir tan tarde como sea posible)**

5. **Entregar tan rápido como sea posible**
6. **Respetar a las personas:** reconociendo que todos queremos trabajar en productos de éxito. Desarrollando buenos líderes capaces de motivar a los equipos de trabajo, practicando y transmitiendo respeto y consideración por los demás. Dotando a las personas del conocimiento necesario para cumplir sus objetivos, estableciendo metas razonables, que puedan alcanzarse y que permitan a las personas auto-organizarse para conseguirlas.
7. **Optimizar el conjunto (visión global)**

Por último, aunque sin una referencia explícita, podemos ver el calado de la filosofía Lean en una de las empresas líderes del sector, si revisamos los 10 principios básicos de Google:

Céntrate en el usuario y todo lo demás llegará
La satisfacción del cliente es la prioridad, y el compromiso por el valor lo manifiestan en una frase que acompaña a este principio: “lo más importante es garantizar que nuestro trabajo sea útil para ti, no para nuestros propios beneficios u objetivos internos.”

Es mejor especializarse en algo y hacerlo realmente bien
El espíritu de mejora continua, fundamental para sostener su éxito: "nosotros nos dedicamos a la búsqueda. Disponemos de uno de los grupos de investigación más grandes del mundo que se dedica exclusivamente a resolver problemas de búsqueda, así que sabemos lo que hacemos bien y cómo podemos mejorarlo.”

La velocidad es un valor seguro
El resultado de una búsqueda es el valor ofrecido al cliente y en cuanto al plazo en entregarlo afirman: "Puede que seamos las únicas personas del mundo que podemos decir que nuestro objetivo es que los usuarios dejen nuestra página principal lo más rápido posible." De nuevo, en este principio, hacen una referencia al espíritu de mejora continua: "seguimos trabajando para que todo vaya incluso más rápido.”

La democracia en la Web funciona
Otro de los valores que debe ofrecer la búsqueda en términos del usuario es la calidad de los resultados mostrados: “La Búsqueda de Google funciona porque se basa en los millones de personas que publican enlaces en sitios web para ayudar a determinar qué otros sitios ofrecen contenido importante.” En este principio, además, se hace una referencia a la importancia de las personas en la organización: “Del mismo modo, desempeñamos un papel activo en el desarrollo de software libre, en el que la innovación surge a través del trabajo conjunto de muchos programadores.”

No tienes que estar en tu despacho para obtener la respuesta que necesitas
Otra vez más, se centran en ofrecer valor al usuario: “El mundo es cada vez más móvil: la gente quiere acceder a la información en cualquier lugar y en cualquier momento. Somos pioneros en el desarrollo de nuevas tecnologías y ofrecemos soluciones para servicios móviles.”. Además, se expresa una preocupación por el beneficio de los colaboradores con la organización: (Refiriéndose a Android) “Esta plataforma móvil no solo beneficia a los consumidores, que disponen de más posibilidades de elección y de nuevas experiencias móviles innovadoras, sino que aumenta las oportunidades de ingresos para proveedores, fabricantes y desarrolladores.”

Se pueden obtener ingresos actuando de forma ética
Justifican la inclusión de la publicidad como un valor añadido: “estamos convencidos de que los anuncios pueden proporcionar información de utilidad única y exclusivamente si están relacionados con lo que el usuario desea encontrar.” Y existe una preocupación ara con aquellos usuarios que no la consideran como una ventaja: “Consideramos que la publicidad puede ser eficaz sin ser molesta.” Y su preocupación por la confianza de sus clientes: “Nuestros usuarios confían en nuestra objetividad, y las ganancias a corto plazo nunca podrían justificar que se quebrantase esa confianza.”

La información no se acaba
De nuevo una referencia a la mejora continua: “nuestros investigadores siguen buscando formas de ofrecer toda la información del mundo a las personas que buscan respuestas.”

La necesidad de información supera todas las fronteras
Nuevos servicios para cubrir nuevas necesidades de los clientes: “Mediante nuestras herramientas de traducción, la gente puede descubrir contenido escrito en el otro lado del mundo en un idioma que no conoce.”

Es posible ser profesional sin llevar traje.
Principio que da el protagonismo a los empleados: “Nuestros fundadores crearon Google en base a la idea de que el trabajo debe ser un desafío y el desafío, una diversión. Pensamos que la genialidad y la creatividad se pueden dar con más frecuencia en una cultura empresarial adecuada”. Fomentando el trabajo en equipo por encima de individualismos: “Hacemos especial hincapié en los logros de equipo y nos enorgullecemos de los éxitos individuales que contribuyen a nuestro éxito global.”. Practicando el *empowerment*: “Depositamos una gran confianza en nuestros empleados.”

Afirman asimismo: “Nuestro ambiente puede resultar informal, pero una vez que las ideas surgen en una cafetería, en una reunión de equipo o en el gimnasio, las comentamos, las analizamos y las ponemos en práctica a una velocidad de vértigo.”

No nos conformamos con unos resultados excelentes
Este principio habla por sí solo, además, expresan su interés por asumir nuevos retos: “Consideramos que ser muy bueno en algo es solo el punto de partida, no el punto final. Nosotros mismos nos fijamos objetivos que sabemos que aún no podemos alcanzar, ya que somos conscientes de que si nos obligamos a conseguirlos, podemos obtener mejores resultados que los esperados.” Todo esto en un contexto centrado en dar valor al usuario final de sus productos, para sus necesidades presentes y futuras: “Intentamos prever las necesidades que todavía no han manifestado nuestros usuarios de todo el mundo y satisfacerlas con productos y servicios que establezcan nuevos estándares.”

4.2. Personas y equipos de trabajo

El enfoque hacia las personas no pasa desapercibido en la literatura, así Poppendieck destaca cómo el componente humano es la piedra angular del éxito en la aplicación de Lean. En este ámbito, propone una serie de herramientas:

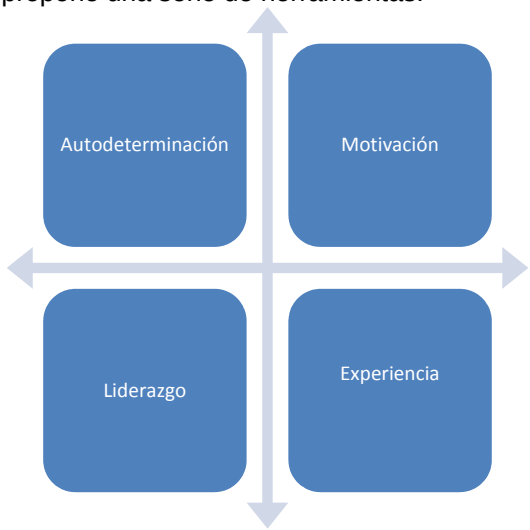

Figura 10 – Herramientas necesarias para el trabajo en equipo

Autodeterminación: basar la transformación en transferir las prácticas en lugar de los principios sobre los que se fundamentan es un error, aunque se consiga realizar la implantación del cambio es difícil mantener esta transformación. En este sentido, la prioridad es conseguir que las personas crean en el cambio y participen de él.

Motivación: el primer elemento motivacional es dar un propósito al trabajo de las personas, por encima de un simple conjunto de tareas, deben entender el propósito de su trabajo, de una forma clara, convincente y alcanzable. Cuestiones como facilitar la comunicación entre el equipo y el cliente, dejar al equipo alcanzar sus propios compromisos ayudan a dar sentido al trabajo que se está realizando. En estas circunstancias, la gestión se convierte principalmente en mantener el momentum, ser un facilitador, detectar las necesidades del equipo para realizar correctamente su trabajo.
Los sentidos de pertenencia a un grupo, de confianza, de competencia y del progreso deben estar presentes para conseguir un equipo motivado. Es destacable la referencia en el apartado motivacional que realiza Poppendieck a la consideración por la conciliación de la vida familiar y laboral, para lo cual recomienda la moderación frente al heroísmo y reservar los esfuerzos adicionales para las emergencias.

Liderazgo: se buscan líderes más que simples gestores, que hagan frente al cambio, marcando el camino a seguir, alineando y motivando al equipo. Un liderazgo basado en el respeto del equipo hacia el líder, por su profundo conocimiento del cliente y de los aspectos técnicos, más allá de una autoridad concedida.

Experiencia: facilitar que los equipos adquieran y compartan su experiencia, que experimenten de manera autónoma, tolerando los errores durante el proceso de aprendizaje y fomentando la transmisión del conocimiento, especialmente el tácito. Por otra parte, la Ingeniería del Software ha de lidiar con multitud de áreas de conocimiento especializado, tanto a nivel técnico como a nivel de negocio, este último tan o más importante como el primero, dado que para conseguir dar valor al cliente es necesario conocer primero su negocio en general y sus necesidades concretas en particular. Por esta cuestión es importante facilitar la creación de comunidades de expertos, especialmente en aquellas áreas críticas para el éxito de nuestra organización. Estas comunidades, además, deben jugar un papel importante en cuanto a la decisión de los estándares a seguir en el área de experiencia que cubren.

Uno de los factores más importantes a tener en cuenta es la necesidad de crear equipos de trabajo, en lugar de grupos. El desarrollo de software implica la resolución de problemas a diario, decisiones complejas que afectan más allá del trabajo de la persona que las realiza. En este sentido, disponer de un conjunto de personas cohesionadas cuya experiencia y conocimientos aporte nuevas perspectivas a los problemas es una baza fundamental para obtener buenos resultados. Para crear estos equipos es necesario darles retos y metas comunes. Los miembros que los componen dependen unos de otros y por tanto debe existir el compromiso del trabajo conjunto para el logro de la meta común.

Algunos factores que dificultan la existencia de equipos de trabajo son los siguientes:

Figura 11 – Factores que dificultan el trabajo en equipo

Individualismo. Las expectativas y la medida del desempeño en la organización que promueven los logros individuales frente a los colectivos.

Exceso de rotación. Al romper los equipos con demasiada frecuencia se pierde la noción de pertenencia, se acaban formando grupos y no equipos.

Pertenencia a varios equipos. Pertenecer a un equipo supone adquirir un compromiso con las personas que lo conforman, pertenecer a varios equipos puede poner estos compromisos en conflicto, a lo sumo se podría formar parte de dos.

Otras dificultades con las que podemos encontrarnos al formar equipos de trabajo están relacionadas con los "especialistas", personas con un conocimiento muy específico en un determinado área.
Si el área de experiencia no es necesaria en todas las etapas del proyecto o la intensidad de la participación requerida es baja, se complica la pertenencia a un equipo. Para mitigar este tipo de situaciones se puede tratar de ampliar sus habilidades o conocimientos hacia otras áreas que faciliten su integración en un equipo. Otra cuestión a tener en cuenta es la tendencia a crear sus propias parcelas, que pueden dejar en un segundo plano los intereses del grupo (por ejemplo, un diseñador gráfico puede realizar un diseño vistoso aunque complejo, haciendo lucir su trabajo dejando a un lado los problemas de la construcción e integración del mismo en el sistema). Una posible solución es poner de manifiesto este conflicto con un estudio del flujo de valor del ámbito en el que se produce.

4.2.1 Incentivos

Cabe distinguir entre dos tipos de organizaciones:

- Las que basan su enfoque exclusivamente en su actividad económica, producir el máximo con los mínimos recursos y generar riqueza para sus gestores y los inversores. En éstas, el trabajador entrega una destreza o habilidad a cambio de una remuneración económica.
- Y aquellas cuyo enfoque es permanecer en el negocio y dar trabajo a largo plazo. En estas, el trabajador entrega su atención y compromiso y la empresa tratará de desarrollar al su potencial individual al máximo, hay un compromiso de ida y vuelta, más acordes con la filosofía Lean.

La evaluación del desempeño del trabajador (típicamente anual) debe ser un momento de reflexión acerca del desarrollo del potencial del trabajador, donde se establecen las actuaciones para conseguirlo, los proyectos en los que participará y la formación que recibirá.

En cuanto al sistema de promociones o compensaciones, se propone seguir una serie de directrices. Así, por ejemplo, el sistema de ser intachable, la promoción entre categorías debe ser transparente y percibida como justa, basada en el trabajo y los méritos. Las recompensas deben basarse en el ámbito de influencia y no en el de control y encontrar mejores elementos de motivación que el dinero (el reconocimiento, la promoción, etc.), dado que su efecto no es sostenible a largo plazo.

4.3. Mejora continua – Kaizen

F. Huda y D. Preston en 1992 publican la primera referencia encontrada que reconoce la aplicabilidad de Kaizen a la Ingeniería del Software, entendido como la mejora continua dentro de un entorno estable de producción.

El papel del componente humano es crítico en el desarrollo de software y Kaizen reconoce esta circunstancia y saca partido de ella: cambia la perspectiva de la resolución de problemas y lo centra en las personas, en buscar soluciones en lugar de culpables y fomentando la búsqueda de los intereses colectivos en lugar de intereses individuales.

En el artículo de Huda y Preston se expone cómo es posible la extrapolación de algunas de las experiencias realizadas en otras disciplinas con esta filosofía a algunos de los problemas de la industria del Software. Las conclusiones más relevantes que destacan son las siguientes:

a. Se pone de manifiesto que su punto fuerte es que gracias a su orientación a las personas y a los procesos, las mejoras resultado de su aplicación se incorporan en el sistema a través de las personas y mediante procesos de control.

b. Se identifica la necesidad de capacitar a las personas y fomentar su participación e involucración en la organización, así como a necesidad prestar atención al componente motivacional de las personas que realizan el trabajo.

Poppendieck propone el uso de **eventos kaizen** para la resolución de problemas. Dado un problema crítico, bien definido, los eventos Kaizen son una herramienta que reúne en un equipo a los representantes de las distintas áreas involucradas en este problema para su resolución. Durante un espacio de tiempo de no más de una semana, trabajarán intensivamente en realizar las modificaciones necesarias en los procesos para solucionarlo. Tras el evento Kaizen, el equipo se disuelve y sus miembros vuelven a realizar sus labores habituales, y el proceso queda implantado con el cambio.

Existen cuestiones relacionadas con Kaizen aplicables en el ámbito del SaaS, cómo los automatismos referidos anteriormente como de obtención de feedback acerca de la ejecución del software, bien implementados, aportan información única para mejorar el sistema en su conjunto. Pero el disponer de esta información no es suficiente, Benefield apunta cómo es necesario fomentar la experimentación continua y disponer de una visión global del producto y de la organización en torno al mismo para lograr un alto rendimiento en el SaaS.

La filosofía kaizen, la proactividad en la búsqueda de la mejora continua a través de todos los aspectos de la vida, sin prejuicios ni culpables es crítica en este contexto: todo el personal involucrado en el proyecto (desarrolladores, clientes, usuarios, etc.) deben sentirse en confianza para expresar su punto de vista y poder participar en el proceso de mejora continua, disponiendo de los canales adecuados para comunicarse.

La gran ventaja del modelo SaaS es que permite la observación sin obstrucciones del uso de los servicios, qué funcionalidades son más o menos populares, el uso que hace el usuario del sistema y el rendimiento del mismo en consecuencia.

Del mismo modo, se pueden plantear mecanismos que permitan realizar pruebas en producción de un conjunto nuevo (o modificado) de funcionalidad para estudiar su éxito, disponiendo de un feedback prácticamente inmediato.

Finalmente, señala el autor, la importancia del alineamiento de toda la organización hacia esta filosofía, en cuanto a la puesta a disposición de los medios para esta experimentación constructiva y sin prejuicios y la constancia de que no todos los experimentos pueden ser un éxito.

4.4. Eliminar el desperdicio

El primer paso para comenzar a aplicar la filosofía Lean es aprender a reconocer los desperdicios. Por esta cuestión, en este apartado se comienza mostrando una equivalencia en el ámbito de la Ingeniería informática de los siete tipos de muda identificadas por Shingeo Shingo en manufactura.
Posteriormente, se trata la adaptación de la técnica 5s en Ingeniería Informática, a un nivel general y particularizando para los desarrollos en lenguaje Java.
Por último, analiza una de las prácticas recomendadas para evitar desperdicio: escribir menos código.

4.4.1. Las 7 mudas

Las mudas expuestas en la filosofía Lean, tienen su equivalente en el ámbito de la Ingeniería del Software, se indican en la siguiente tabla:

Contexto manufactura	Contexto ing. Software
Inventario	Trabajo sin terminar
Sobreproducción	Características "extra"
Sobre-procesamiento	Reaprendizaje
Transporte innecesario	Cambio de persona asignada a una tarea
Movimientos innecesarios	Cambio de tarea
Esperas	Retrasos
Defectos	Defectos

Tabla 2. Las 7 mudas en el ámbito de la ingeniería del software

Inventario – Trabajo sin terminar

Es una de las fuentes de desperdicio más peligrosas, puesto que un trabajo no se considera como terminado hasta que no se encuentra en el sistema en producción y, si no llega a estarlo, puede llegar a suponer la pérdida de toda la inversión realizada.

Ejemplos de trabajo sin terminar

- **Documentación sin codificar:** Documentos de requisitos y diseño técnico que aún no se comienzan a implementar
- **Código sin consolidar:** este concepto guarda relación con las herramientas de control de versiones. Deben minimizarse situaciones como código sin subir al repositorio o desarrollos paralelos sobre un mismo recurso que requieren a posteriori un proceso de mezcla
- **Código sin probar**
- **Código sin documentar** (en la medida que requiere)
- **Código sin desplegar:** los resultados han de ponerse en producción evitando demoras

La forma de evitar este inventario es dividir el trabajo en lotes pequeños o iteraciones.

Sobre-procesamiento – Procesos "extra", reaprendizaje
Procesos "extra" como documentación innecesaria del Software: han de evitarse los documentos que no van a utilizarse.
El reaprendizaje, por una incorrecta gestión del conocimiento se deriva en rehacer trabajo, "reinventar la rueda".
Las siguientes cuestiones inciden en el coste añadido por el reaprendizaje:

Código sin documentar: la documentación justa y necesaria en el momento del desarrollo evita el coste de reaprendizaje en una modificación posterior del mismo.

Planificación deficiente: si la asignación de tareas a cada miembro del equipo de trabajo no toma en consideración su conocimiento previo de las mismas se dan situaciones de reaprendizaje, por ejemplo asignando a un programador A una tarea codificada por un programador B y viceversa, ambos han de aprender qué realizó su compañero con el consiguiente coste. En determinadas situaciones este tipo de asignación puede ser una buena práctica, normalmente en partes críticas de una aplicación, pero ha de realizarse bajo una especial supervisión.

Calidad deficiente: si un error llega hasta el entorno de producción, puede incrementar la dificultad para localizar su causa e implicar el reaprendizaje incluso si el desarrollador que aborda la corrección es el mismo que realizó el código que lo provoca.

Tareas en paralelo, o excesivo cambio de tarea: cada vez que se retoma una tarea hay un coste de este tipo.

Comunicación / gestión del conocimiento deficiente: si bien cada vez existen más herramientas para facilitar estas cuestiones, sistemas para compartir el conocimiento, mejores herramientas de búsqueda de información, wikis, etc.

La captura efectiva del conocimiento en una organización tiene una dificultad implícita, que pone en evidencia el fuerte contraste del punto de vista occidental frente al oriental. Así, mientras el primero considera que este conocimiento es algo que puede y debe ser escrito, el último considera el conocimiento tácito, que viene de la experiencia y no el estudio, algo no fácilmente almacenable o procesable por un ordenador y difícil de formalizar y transmitir.

Para facilitar la transmisión del conocimiento, es posible utilizar el informe A3 de Toyota: la condensación de las ideas, que deben exponerse de manera resuelta en una hoja de tamaño A3, y con seis directrices fundamentales a seguir:

- Usar las mínimas palabras posibles
- Preferentemente utilizar gráficos, diagramas y tablas mejor que palabras.
- Todo debe caber en una cara A3
- Debe ser un documento dinámico, usarse y modificarse según los comentarios obtenido al mismo.
- Si no se ajusta a un A3, condensarlo en un A4
- Su contenido y estructura han de servir a su propósito

Es posible pensar en varios propósitos para esta filosofía de informe, así, por ejemplo, para la resolución de problemas, podríamos contar con los siguientes elementos a incorporar al A3: resumen del problema, análisis causa efecto, solución propuesta, experimentos previstos, mediciones y comentarios.

Sobreproducción – Características "extra"
Cada línea de código debe responder a una característica que se sabe con certeza que es necesaria y en ese momento, no solo teniendo en cuenta el coste de su producción, sino también el de su mantenimiento en cuanto a tiempo de compilación, posibles errores y complejidad añadida al código en su conjunto.

La inversión realizada en *frameworks* en busca del beneficio de las economías de escala también puede suponer desperdicio. Si no se acaban utilizando, este tipo de adquisiciones (ej. compra a un tercero de una librería para la representación gráfica) o desarrollos dentro de la propia organización no se amortizan en un escenario de baja reutilización. Ponen como ejemplo el modelo de desarrollo en cascada, donde el Ingeniero de Software ha de ir más allá de las necesidades inmediatas del proyecto con un pensamiento a largo plazo que incrementa el riesgo de implementar requisitos que nunca se utilizarán.

Transporte innecesario - Cambio de tarea, Cambio de recurso
Cada vez que el código pasa de una mano a otra, existe riesgo de coste añadido por la transmisión del conocimiento.
La asignación de personas a más de un proyecto o tareas paralelamente también induce este tipo de desperdicio, dado que necesitarán emplear un tiempo para concentrarse en una tarea antes de comenzar a trabajar en ella, además de las posibles pérdidas por interrupciones en el trabajo (por ejemplo, la resolución de dudas).
Como norma general, la forma más eficiente de acometer varias tareas que comparten el mismo equipo de trabajo y deben finalizar dentro de un mismo plazo es acometerlas secuencialmente y nunca en paralelo para evitar desperdicios.

Por otra parte, han de evitarse las reasignaciones de tareas incompletas. El paso de persona en persona provoca pérdida de tiempo y conocimiento. De no ser posible, ha de potenciarse la comunicación que favorezca y agilice la transmisión del conocimiento tácito.
Otro problema es el derivado de la entrega al cliente de un nuevo requisito que, de no estar bien documentado, provocará su llamada para pedir soporte.

Existen una serie de consideraciones a tener en cuenta para reducir el impacto de los cambios de asignación, aparte de tratar de minimizar estos cambios en la medida de lo posible:

- Formar equipos multidisciplinares que permitan la formación dentro del propio equipo.
- Comunicación fluida: fomentar la comunicación cara a cara, la observación directa, la interacción con prototipos.
- Mostrar avances del trabajo que se está realizando para obtener feedback tan pronto como sea posible.

En cuanto a los cambios de tarea, disponer de un equipo específico para dar mantenimiento correctivo reduce el número de cambio de tareas, así como otras estrategias basadas en agrupar este tipo de actuaciones y realizarlas minimizando las interrupciones al resto de desarrollos.

Esperas
Es uno de los mayores ladrones de tiempo en el desarrollo de Software.

Ejemplos de espera
• **Retraso del comienzo del proyecto** • **Retrasos en la asignación de recursos al proyecto** • **Retrasos en revisiones y validaciones** • **Retrasos en las pruebas** • **Retrasos en las implantaciones**

Estas esperas, además, provocan la reducción del plazo efectivo para producir y entregar el valor al cliente.
Para combatir este tipo de pérdidas, es necesario retrasar las decisiones tanto como sea posible, hasta tener la certeza de que son correctas y que aportarán valor al cliente.

Movimientos innecesarios
Se consideran los movimientos de los miembros de los equipos de trabajo para cuestiones como resolver dudas o realizar reuniones. Además, hay que considerar como movimientos la dificultad para localizar determinados artefactos necesarios para realizar el trabajo: documentos, fragmentos de código, librerías, etc.

Defectos
El efecto de los mismos es factor del impacto del mismo y del tiempo que tarde en detectarse: cuanto antes se detecte, menor será. La práctica recomendada para paliar este tipo de muda es probar inmediatamente, integrar el código frecuentemente y actualizar el sistema en producción a la mayor brevedad posible.

Por otra parte, se hace referencia en la bibliografía al uso del VSM, observar las actividades por las que pasa el producto software con una óptica que facilite la eliminación de las causas raíz de los desperdicios. Esta técnica se tratará con más detalle en un apartado a tal efecto.

Actividades de Gestión
Las actividades de gestión, si bien no aporta directamente valor al producto, tienen un gran impacto en el desperdicio de una organización. Así, por ejemplo, generan desperdicio una mala priorización de las tareas a realizar por el equipo o una incorrecta liberación de los requisitos implementados que provoquen el aumento del inventario y los tiempos de espera.
Igualmente señala que los sistemas de control y seguimiento de proyecto deben mantenerse simples y su complejidad puede ser un indicador de otros desperdicios en la cadena de producción.

4.4.2. Clave: escribir menos código

Una de las potenciales fuentes de desperdicio es la complejidad. La complejidad puede dificultad el cambio, hacerlo más lento y menos seguro.
Para esta cuestión se ha examinado el caso de éxito de Zara, cuyo gasto en TI es del 0.5% de sus ingresos y cuál es su enfoque respecto a la tecnología:

- **Es una ayuda para la toma de decisiones**: no sustituye el juicio de las personas, les provee de información y sugerencias.
- **La informatización es dirigida y estandarizada**: se debe utilizar únicamente la solución corporativa y la tendencia en la evolución de la misma es minimizar sus características en lugar de maximizarlas.
- **Las iniciativas tecnológicas provienen de dentro de la**
- **organización**: TI soluciona los problemas del negocio y no al revés.
- **Enfoque en el proceso**
- **Existe un fuerte alineamiento TI – Negocio**: ambos mundos se conocen y entienden.

El coste de la complejidad es exponencial y a menudo se convierte en el coste dominante. Por esta cuestión, el código debe procurarse pequeño, simple y limpio y se dan una serie de directrices:

- **Evitar la complejidad**: El factor más determinante es la complejidad del mismo, dificulta y encarece el mantenimiento y los cambios tienden a presentar más errores. El coste de la complejidad crece exponencialmente con el tiempo.
- **Justificar cada característica a implementar**: en caso de software comercial debe responder a una necesidad del mercado y en proyectos a medida ajustarse a las necesidades expresadas por el cliente.

- **Conjuntos mínimo de características útiles:** dividir el software en conjuntos reducidos de funcionalidad según este criterio y cometerlas y ponerlas en producción por orden de importancia, permitiendo al cliente disponer del incremento del valor que ofrece el producto software en plazos más cortos.
- **No automatizar la complejidad:** el paso previo a la automatización de un procedimiento es su simplificación.

4.5. Heijunka: Producción nivelada

El término Heijunka está presente en las metodologías ágiles: tienen preferencia por la polivalencia de las personas para realizar distintos tipos de tareas. Así, a mayor polivalencia de los miembros del equipo, más sencillo será nivelar la producción de Software.
Además del uso de una metodología ágil y aprovechar la polivalencia, en este estudio se propone la extrapolación del **método de gestión de almacenes** para el desarrollo de software en el ámbito de proyectos de soporte y mantenimiento, que no suelen ser fácilmente abordables con las metodologías ágiles habituales.

De esta forma, se expone cómo con esta solución es posible nivelar la producción incrementando el rendimiento global. Las prácticas heijunka que se implementan son las siguientes:

- Ejecución del trabajo utilizando el sistema pull del almacén.
- Gestión de la carga de trabajo de una sola cola.
- Gestión visual de la carga de trabajo y control autónomo de la misma.

Las peticiones llegan a un almacén en el que se gestionan mediante una única cola y son procesadas de manera secuencial.

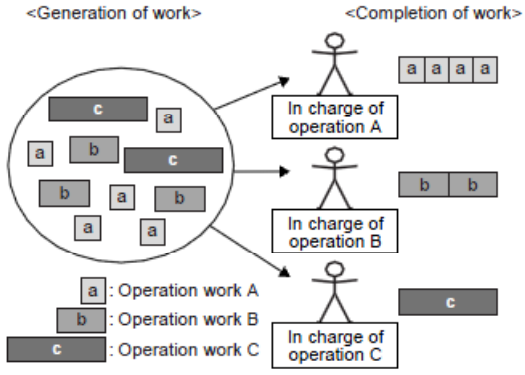

Figura 12 - Esquemático del método de gestión de almacenes

Mediante la gestión visual, los desarrolladores pueden ver el trabajo pendiente en cada una de las operaciones tipo que se contemplen y pudiendo ellos mismos detectar y corregir un desequilibrio entre la carga de trabajo y la capacidad en alguna de ellas de manera autónoma.

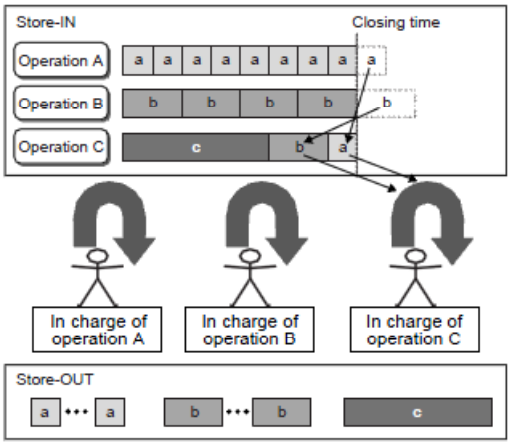

Figura 13 - Método de gestión de almacenes: gestión visual

Una cuestión importante a tener en cuenta, es que cada uno de los miembros del equipo sólo atiende una única petición a la vez (flujo unitario).

4.6. Procesos estables y estandarizados

Scrum y CMMI ofrecen desde distintas perspectivas una solución combinada para la gestionar con éxito la complejidad y la incertidumbre en los requisitos de los proyectos de desarrollo Software: CMMI provee disciplina y estabilidad en los procesos y Scrum mejora la adaptabilidad.
Con la implementación de CMMI se puede mejorar la aproximación ágil mediante la institucionalización, dotando de una mayor disciplina al uso de las prácticas ágiles. Algunas de las actividades a realizar para implementar esta institucionalización son:

- Establecer y mantener una política organizacional para la planificación y realización de prácticas ágiles.
- Formar a las personas para realizar prácticas ágiles.
- Recopilar los resultados del uso de métodos ágiles para su utilización en la mejora continua de los mismos.

De forma similar, Scrum suple algunas de las carencias de CMMI, como por ejemplo el tomar en consideración cuestiones subyacentes en la organización que influyen en el desarrollo de los trabajos (la responsabilidad del scrummaster es resolver cualquier impedimento al transcurso normal de la actividad)
El uso de Lean como paradigma para la mejora continua en este contexto mediante las prácticas recomendadas por Lean Software Development se examina en un ejemplo práctico revisado en este trabajo:

- Reducción del ciclo típico de scrum de 1 mes a únicamente dos semanas.
    - Se redujo el coste de corrección de los defectos por disponer de feedback en menor plazo y aumentó el nivel de satisfacción del cliente.
- Incorporación de técnicas de desarrollo dirigido por pruebas.
    - Aumentó el conocimiento del dominio del negocio en todo el equipo, se mejoró el plazo de finalización y se redujo el número de defectos.
- Detección del valor en términos del cliente, analizando conjuntamente los requisitos expresados por este para determinar su necesidad real.
    - El coste y número de requisitos se redujo a la mitad (en el caso realizado, era un cliente cuyo presupuesto no alcanzaba el coste inicial)

4.7. Gestión visual

La gestión visual es cada vez más frecuente en el ámbito de la Ingeniería del Software. Estos son algunos ejemplos aplicados a distintas áreas:

Ejemplo 1: resultado de la construcción de un conjunto de proyectos en la herramienta de integración continua Hudson.

1. Se utiliza un código de colores para identificar el resultado de la última construcción del sistema: cancelada, correcta o incorrecta.
2. El detalle emergente de un proyecto nos muestra que las nubes y claros son debidas a la pérdida de estabilidad, 1 de las 3 últimas construcciones falló.

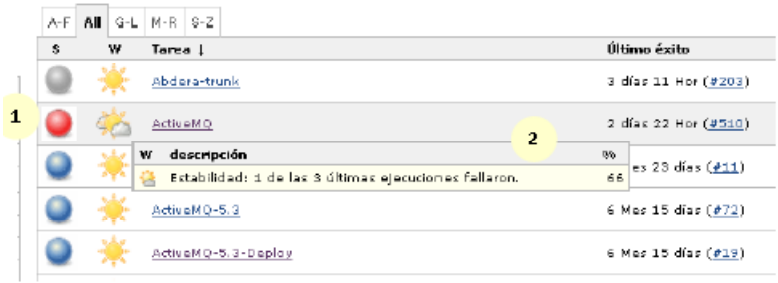

Figura 14 - Gestión visual integración continua 1

Ejemplo 2: resultado de la construcción de un proyecto en la herramienta Hudson.

1. Mediante un código de colores se indica el resultado de la comparación de la última construcción respecto a la anterior, si hubo más errores, sobresaltará.
2. Con un código de colores es posible distinguir los módulos que contienen errores de los que no.

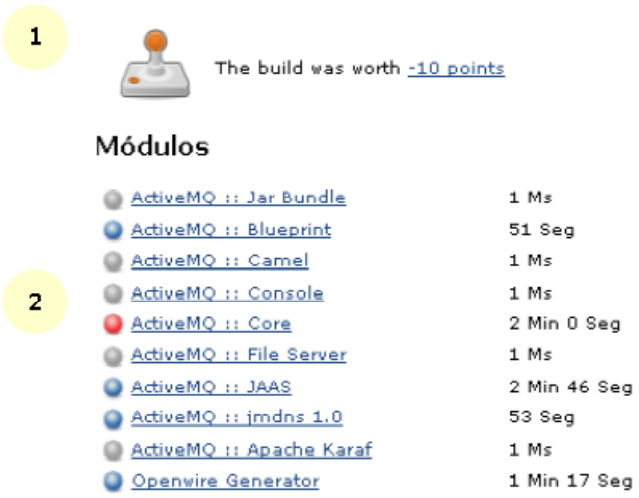

Figura 15 - Gestión visual: integración continua 2

Ejemplo 3: gráfica sobre la relación de incidencias recibidas y realizadas en la herramienta de gestión operativa JIRA

Figura 16 - Gestión visual - herramienta de gestión operativa

Con un simple vistazo podemos ver la relación durante el intervalo entre problemas resueltos y problemas sin resolver.

Ejemplo 4: entorno de desarrollo Eclipse alertando visualmente de la ocurrencia de un error.

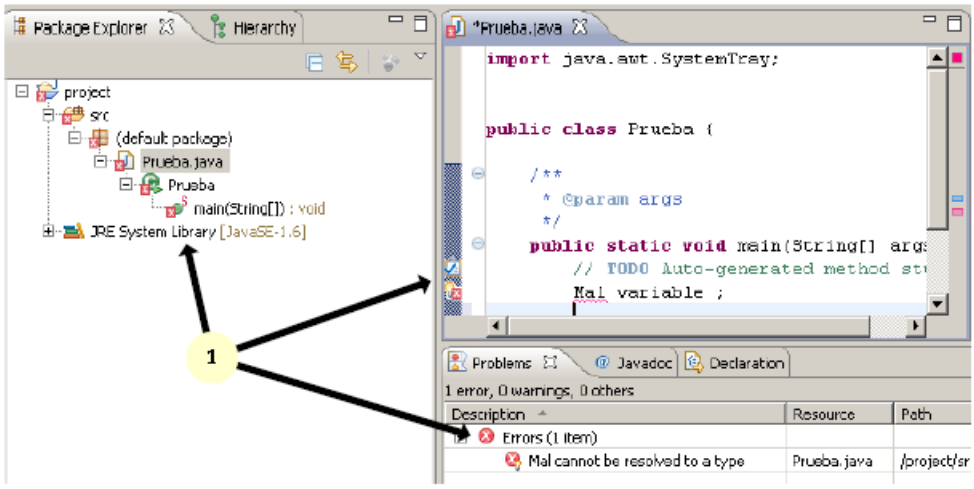

Figura 17 - Gestión visual - entorno de desarrollo

Mediante iconos que se activan automáticamente, el desarrollador detectará la ocurrencia de un error (estático) de programación en el momento que lo está codificando.

4.8. Just in time

Se han encontrado un buen número de conceptos relacionados con el Just in Time aplicados al ámbito de la Ingeniería del Software, tanto en artículos como en libros.

Se incluyen en este apartado diversas estrategias para **reducir el tiempo de ciclo** en proyectos software. A continuación, se examina la implementación del **sistema pull** en este ámbito, mediante una aproximación **Kanban – Scrum**, los conceptos de ambos sistemas se muestran a continuación.

4.8.1. Enfoque ágil versus enfoque tradicional

Existe una creciente necesidad de la aplicación de la filosofía Just In Time en los proyectos Software.
En las metodologías clásicas, una vez establecido el alcance del proyecto se realiza una estimación de esfuerzo, coste y plazo para transformar los requisitos expresados por el cliente en un sistema en producción que les dé respuesta.

La estimación de esfuerzos es una actividad que se ha descrito como “el eslabón más débil de la cadena” de producción de software. La gran incertidumbre que rodea esta actividad y las desviaciones en la ejecución de los proyectos respecto a las previsiones es reconocida y sigue sin solución en la actualidad, no obstante, sigue siendo en muchos casos medida del éxito de un proyecto.
Además se ha de tener en cuenta que más de la mitad de las fuentes de error son debidos a una incorrecta toma de requisitos, dada la frecuencia con que no es posible obtener esta información con suficiente nivel de detalle al comienzo de los proyectos y, cuando se tiene, suele estar sujeta a modificación por cambios (en el negocio, en la

operativa del usuario a la que se pretende dar solución, en el cliente, etc.) de manera directamente proporcional a la duración del desarrollo.

En este contexto de incertidumbre, las relaciones contractuales que tratan de garantizar el cumplimiento de requisitos, alcance, coste y calidad son fruto de un pensamiento erróneo y de la falta inicial de de confianza y colaboración cliente-proveedor, afirma el autor. Así, es fácil caer en una aversión al cambio por anteponerse la prioridad de cumplir con costes y plazos en detrimento de obtener un producto que responde a las necesidades del cliente.

Por estas cuestiones, existen multitud de publicaciones que aseveran que los enfoques tradicionales no son válidos y que la planificación debe ser adaptativa y no predictiva.

El enfoque ágil asume que los requisitos no pueden ser establecidos con seguridad desde el comienzo del proyecto y se acepta el cambio como algo inevitable y cuestiones como el plazo y el coste se pueden establecer antes del alcance, pues se persigue la puesta a disposición del cliente cada cierto tiempo y dentro de un coste de un determinado conjunto de requisitos (los de mayor prioridad). De esta forma, el cliente puede saber cuándo obtendrá un conjunto de requisitos y el proveedor los recursos de los que dispone para producirlos. Las decisiones se toman en torno a la prioridad de los requisitos y el rol del proveedor será asegurar el flujo constante de producto según esta prioridad. Además, parte del conjunto de requisitos no abordados, por tener menor prioridad y no estar en el alcance de la iteración, a menudo acaban considerándose innecesarios. El producto obtenido con este enfoque es más acorde a la filosofía “Lean”.

4.8.2. Tack-time plannig – Reducción del tiempo de ciclo

Takt time en software puede entenderse como el tiempo en que un requisito se ha de poner a disposición del cliente.

Reducir los tiempos de producción o tiempo de ciclo es una ventaja competitiva crítica en los proyectos Software, por las cada vez mayores exigencias de los clientes en cuanto a plazos y por la competencia con otras empresas que evolucionan sus propios productos a contrarreloj, algo especialmente intenso en productos Web. Aunque esta no siempre es la estrategia a seguir, así Poppendieck propone la estimación del **coste del retraso** como herramienta, evaluando en términos económicos el importe a invertir en la mejora del tiempo de ciclo y la ventaja económica que repercute de la misma.

H.T. Yeh expone cuatro estrategias para reducir los tiempos de ciclo en desarrollo Software: tiempo de aprendizaje, tiempo debido al desperdicio, tiempos de espera y tiempos de repetición.

- **Reducir el tiempo de aprendizaje**, capturando el conocimiento del producto. La correcta documentación y el acceso ágil a la misma mejoran este factor. Paquete de “Bienvenida al proyecto” orientado a los miembros que se incorporan al equipo de proyecto para facilitar su aprendizaje. Igualmente la estandarización de herramientas y procedimientos a nivel de organización facilita la reutilización del material destinado al aprendizaje.

- **Reducir el tiempo debido al desperdicio,** una cuestión que se trata con detalle en el apartado "eliminar el desperdicio".
- **Reducir los tiempos de espera**, aquellas vinculadas al cliente (por ejemplo, la demora en la toma de decisiones, en la resolución de dudas sobre requisitos o el retraso en la aceptación), o vinculadas a la tecnología utilizada (por ejemplo, espera a la ejecución del proceso compilación y ensamblaje y despliegue en el entorno de un sistema). Algunas fórmulas para reducir estos tiempos apuntadas por este autor son:

    - **Equipos pequeños con responsabilidad compartida**, involucrados con el proyecto y con un contacto fluido con el cliente y una comprensión mayor de los requisitos del proyecto y los objetivos que persiguen. Una idea en la línea de las metodologías ágiles.
    - **Reducir el tiempo de set-up:** por ejemplo, para la realización de un test de sistema sobre una versión del software es necesario esperar a que se realice su despliegue sobre el entorno de pruebas.

- **Reducir los tiempos de repetición**, mediante la reutilización en el amplio sentido: documentación, código, etc. pasando del paradigma de la codificación a la del ensamblaje de componentes, una técnica con otras ventajas adicionales como mayor calidad y funcionalidades más completas (por ejemplo, "displaytag" es un componente Java para mostrar listados Web implementa un elevado número de funcionalidades, paginación, ordenación, exportación de la información a formato CSV, PDF, etc. y para incorporarlo a un proyecto se necesita poco más que una línea de código cada vez que se quiera utilizar)

  Para una mejor gestión de la reutilización, se propone la creación de una infraestructura global de reutilización:

    - **Equipo de reutilización de entornos:** responsable de la provisión y reutilización del hardware y plataformas (sistemas operativos, entornos de desarrollo, etc.)
    - **Equipo de reutilización de componentes:** responsable de la provisión y reutilización de widgets, librerías, plantillas, etc.

  Estos dos equipos tendrían los siguientes cometidos sobre sus áreas de responsabilidad:

    - **Selección:** de los reutilizables necesarios para cada proyecto.
    - **Producción, adquisición, evolución:** de aquellos elementos reutilizables de los que nos se dispone en la organización o cuya funcionalidad sea insuficiente.
    - **Documentación:** práctica que facilite el uso y la transmisión del conocimiento.
    - **Soporte y mantenimiento:** apoyo a los equipos de desarrollo que hacen uso de esta infraestructura, solucionando dudas, realizando correcciones sobre errores detectados en los reutilizables, etc.
    - **Difusión:** de aquella información relevante para los proyectos usuarios de los reutilizables (mejoras, parches, etc.)

Aplicación de la teoría de colas

El uso de la **teoría de colas** es propuesto por Poppendieck para reducir el tiempo de ciclo:

Típicamente existen departamentos dentro de la organización con tendencia a que se conviertan en cuellos de botella y comiencen a formarse colas de trabajo pendiente, por ejemplo el departamento de testing.

Una de las maneras de reducir el tamaño de las colas es reducir el tamaño de los "paquetes" de trabajo, evitando las esperas asociadas a la acumulación del conjunto mínimo de tareas para acometer. En el ejemplo del departamento de testing, una posible estrategia es, en lugar de encomendar la realización de las pruebas de todo el sistema de una vez, irlo solicitando por áreas funcionales.

Otras de las optimizaciones de las colas es eliminar la variabilidad en cuanto al tiempo de procesado, cuestión a la que reducir el tamaño de los paquetes colabora, la otra estrategia aplicable es el procesado en paralelo, incrementar el número de agentes que pueden procesar el paquete en la cola, lo que, además añade tolerancia a fallos en el sistema de producción, si un agente se detiene, la cola no tiene por qué hacerlo.

En el ejemplo, el equipo de testing puede estar compuesto por varios técnicos que atienden indistintamente las peticiones según van llegando).

Por otra parte, si en general se recomienda mover la variabilidad hacia el final del proceso, si este es iterativo (como es el caso de las distintas modalidades de desarrollo iterativo) hay que prestar especial atención a la hora de seguir este tipo de recomendaciones, puesto que no son de aplicación inmediata.

Así, por ejemplo, si las pruebas de aceptación se consideran el final del proceso en un desarrollo en cascada, en un proceso iterativo preceden la siguiente iteración, de forma que un cuello de botella en esta actividad nos detendría el proceso completo.

En general, el exceso de carga de trabajo, complica sobremanera la reducción del tiempo de ciclo. En la Ingeniería del Software, esto no es una excepción, así volviendo al ejemplo, si el equipo de pruebas sufre de exceso de trabajo y el tamaño de la cola asociada al mismo aumenta, como aumenta el tiempo en que el feedback se envía al equipo de desarrollo que solicitó las pruebas. Si persiste esta dinámica, motiva la creación de paquetes de pruebas de mayor tamaño por los equipos de desarrollo y la reducción de la intensidad con las que se realizan las pruebas por el equipo de testing, esta espiral de sucesos se encadena llegando a producir un fuerte impacto en el tiempo de ciclo, que se degrada exponencialmente y con mayor intensidad cuanto mayor es el tamaño del paquete y la carga de trabajo soportada, como se aprecia en la siguiente figura.

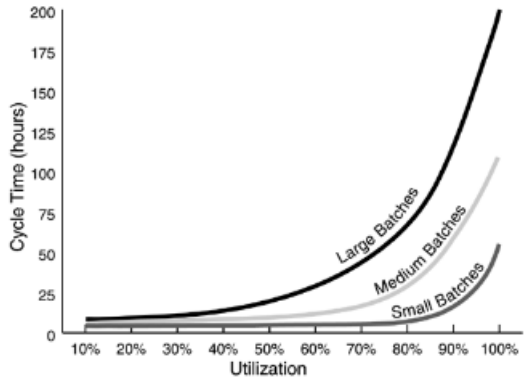

Figura 18 - Impacto del tamaño del lote en el tiempo de ciclo

Por último Poppendieck hace una referencia a la **teoría de restricciones** y el uso iterativo de la estrategia de localizar y solucionar el cuello de botella que limita la capacidad de producción para mejorar gradualmente el desempeño del sistema.

4.8.3. Sistema pull

EL concepto de sistema pull se incorpora en las pruebas de aceptación de las metodologías ágiles. Las pruebas de aceptación verifican que la implementación realizada es acorde a los requisitos expresados por el cliente. En las metodologías ágiles una técnica habitual es que las pruebas conduzcan la implementación (como se expuso en *XP*), por el contrario, en las tradicionales la especificación se obtiene en procesos previos a la implementación.

Por otra parte, uno de los requisitos para agilizar el proceso de desarrollo es que el equipo de trabajo sepa qué debe hacer, bien porque explícitamente se les indique o disponer los medios para que puedan descubrirlo por sí mismo, siendo esta última opción la que ofrece mejores resultados. La ingeniería del software es una disciplina que implica una continua resolución de problemas, por esta cuestión las personas deben disponer de la autonomía suficiente para tomar sus propias decisiones y actuar sin esperas innecesarias.

En un entorno que requiera velocidad no hay tiempo para que la información suba por la cadena de mando y baje en forma de directrices constantemente, si bien es necesario disponer de mecanismos que permitan la coordinación del trabajo. Una de las claves para conseguirlo es utilizar un sistema pull para atender las necesidades del cliente.

Para implementar un sistema pull en desarrollo software Poppendieck propone una solución mixta Kanban – Scrum. El desarrollo se realizará mediante un conjunto de iteraciones cortas (máximo un mes) que darán respuesta a los requisitos expresados por el cliente al comienzo de cada iteración.

Cada uno de estos requisitos se identifican por un nombre y una descripción más o menos somera del mismo (no llega a ser la especificación detallada del mismo), la ficha donde se recoge esta información es en una ficha denominada “Story Card” en la

terminología ágil, requisitos que han de ser implementados (típicamente en pocos días). Estas fichas se enriquecen con una estimación acerca del tiempo necesario para implementar el requisito (lo dan los desarrolladores) y la prioridad del mismo (la asigna el cliente).

En cada comienzo de iteración, se elije el conjunto de fichas que se abordarán (atendiendo al tiempo de la iteración, el equipo de trabajo disponible y la duración estimada de las fichas y su prioridad). Las fichas elegidas, pasan a denominarse fichas "Kanban". Estas fichas se disponen en un tablero, en un área denominada "Por hacer" (To Do).
Estas fichas no son asignadas a los desarrolladores para su realización, serán ellos los que elegirán las fichas en las que quieren trabajar (el trabajo en esta aproximación es autodirigido), añaden su nombre en la ficha, y las colocarán en un área distinta "En desarrollo" "Checked out"). Una vez terminadas las pruebas correctamente, se pasarán a un área "Pruebas superadas".

El estado de la producción en esta aproximación se puede comprobar echando un vistazo a este tablero, al tiempo que los desarrolladores saben la tarea que han de realizar y de cuánto tiempo disponen y, cuando acaban, cuál sigue.

Para algunas cuestiones, como por ejemplo, que el nivel de definición de los requisitos en las fichas no sea suficiente para los desarrolladores, se realiza una reunión periódica, típicamente diaria que debe ser breve (15 minutos) y debe asistir todo el equipo. Las cuestiones a tratar en el este espacio de tiempo son: cada miembro, expone qué hizo el día anterior, qué deben hacer el día de la reunión y si necesitan ayuda para hacer su labor. La función del líder en esta reunión es interferir por el equipo, así si un desarrollador necesita información adicional del cliente para un requisito, el líder debe proveer un canal de comunicación para que la reciba (el cliente o un agente representativo del mismo) Si alguna cuestión requiere de una discusión que vaya a dilatar la reunión ha de posponerse para una reunión que sólo implique a las partes afectadas en su resolución.
Decir que el tiempo de cada iteración no ha de superar típicamente un mes, para evitar que se acumulen fichas sin realizar o no tengan suficiente nivel de detalle para la sostenibilidad del sistema pull.
En el éxito de este sistema es fundamental el uso del control visual como habilitador del trabajo autodirigido. Esta necesidad del sistema pull necesita de información visible, accesible y actualizada del proceso. Se menciona el concepto en Ingeniería del Software de los "radiadores de información" referido a los elementos visuales en este ámbito, para la gestión visual y que hagan los problemas visibles (ej. fichas Kanban, listas de problemas, listas de mejoras, candidatos a refactorizar, resultados de los test, etc.)

4.8.4. Kanban - Scrum

Ambas herramientas de proceso, dan una serie de directrices para realizar el trabajo de manera más eficaz, siendo Scrum más prescriptivo que Kanban. En el siguiente gráfico podemos apreciar el nivel de prescripción de algunas metodologías ágiles frente a Kanban en cuanto a la cantidad de prescripciones que realizan.

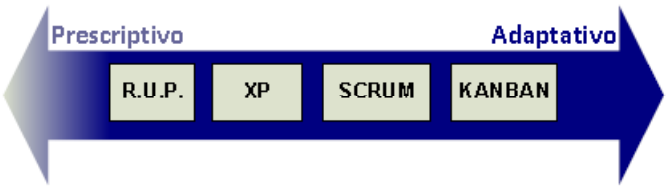

Figura 19 – Kanban y ágiles: Prescripción vs adaptabilidad

En cuanto a la limitación del trabajo en curso (WIP). Scrum limita implícitamente el trabajo en curso por iteración, mientras que Kanban lo haría por estado dentro del proceso de desarrollo.

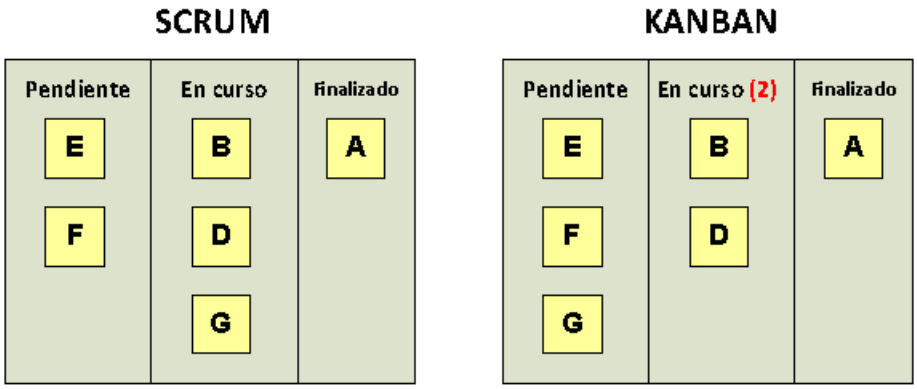

Figura 20 - Tableros básicos: Scrum frente a a Kanban

Así, en el ejemplo, vemos como con la solución SCRUM, nada impide que se comiencen todas las tareas y pasen al estado “en curso” mientras que en la solución KANBAN no puede haber más de dos en esta situación. Esto es, mientras en Scrum el WIP se limita por unidad de tiempo, en Kanban esta limitación vendrá dada por el estado del flujo de trabajo.

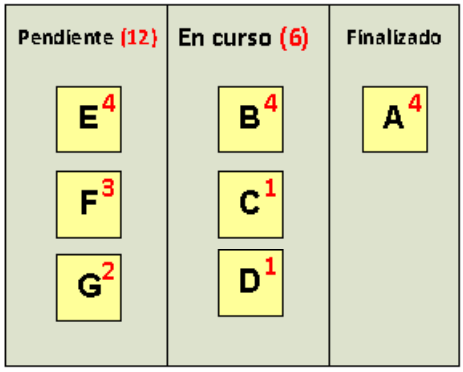

Figura 21 - Tablero Kanban: limitación por estado

La limitación debe extenderse a todos los estados de la cadena de valor. Una vez hecho esto, se facilita la medición y predicción de los tiempos que necesita cada elemento en su

paso por los distintos estados, permitiendo hacer planes de entrega más realistas. Para esta cuestión es necesario que estos elementos sean de un tamaño similar. Si consideramos este tamaño como el número de horas estimadas de trabajo, podemos tomar alternativamente como la limitación del WIP, por ejemplo, como una cantidad de horas estimada máxima.

Tanto Scrum como Kanban se basan en la experiencia, es necesario experimentar con el proceso y adaptarlos. Ninguno de estos sistemas proporcionan las respuestas por sí solos, sus reglas y limitaciones son una guía para la mejora de los procesos. El elemento más crítico es precisamente la retroalimentación, el feedback necesario para adaptar el proceso y mejorarlo: cuanto más rápido obtengamos esta información, mejor.

La combinación de Scrum + XP proporciona feedback a varios niveles, desde errores detectados en el mismo momento que se producen, gracias a la programación en parejas, como la información que recibimos después de cada sprint.

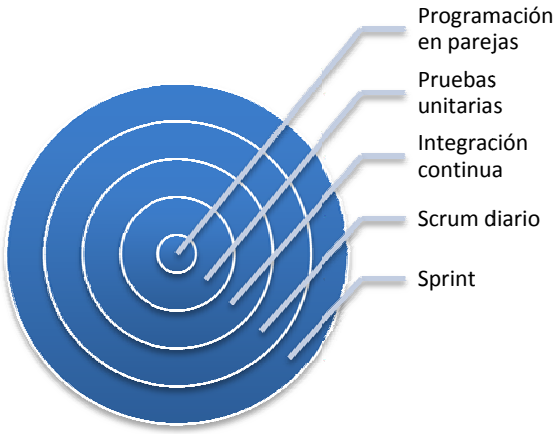

Figura 22 - Niveles de feedback Scrum + XP

Kanban da información visual en tiempo real, de forma que podemos identificar cuellos de botella, trabajo acumulado en un estado y un espacio vacío en el estado posterior. Además, el Lead Time medio, cada vez que un elemento sale del proceso completo.
Scrum se resiste a los cambios durante la iteración. El compromiso se adquiere al comienzo del sprint y si un requisito llega se añade a la pila de producto y al menos ha de esperar hasta el siguiente sprint para ser desarrollado y entregado. La ventaja del uso de Kanban frente a esta limitación en cuanto al tiempo de entrega es manifiesta. Otra ventaja importante que aporta Kanban frente a Scrum es elimina la necesidad de equipos multidisciplinares y un mismo flujo de trabajo puede ser compartido por varios equipos.

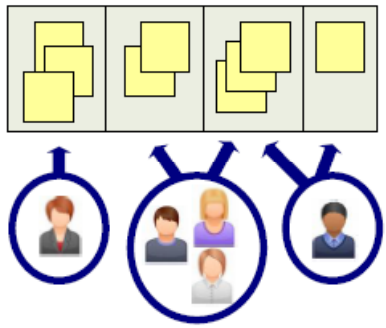

Figura 23 - Tablero Kanban compartido por varios equipos

La limitación temporal del sprint de Scrum es una restricción que no se presenta en la aproximación con Kanban. Si como norma general es recomendable que el tamaño de cada elemento no sea excesivo, Kanban solo establece la limitación del WIP.

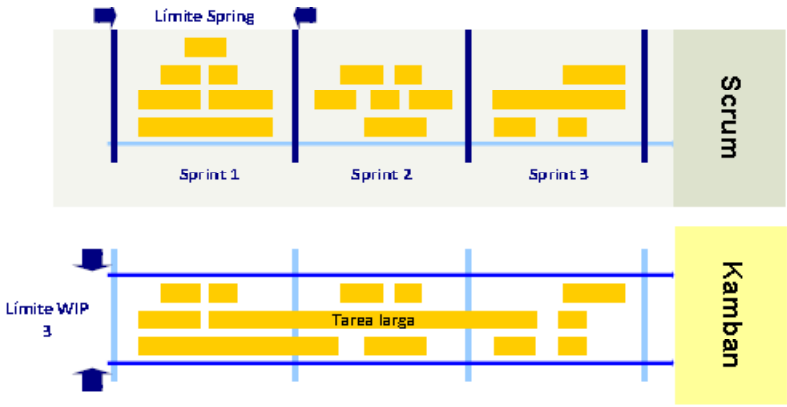

Figura 24 - Limitación del WIP en Scrum frente a Kanban

4.8.5. Quick Changeover

Un ejemplo de tiempo de Setup aplicado a esta disciplina es el necesario para la compilación y ensamblado del código.
En un proyecto de tamaño medio, este proceso afectará a miles de ficheros de código fuente, haciéndolo costoso en tiempo. Por esta cuestión, se puede caer en la tentación de agrupar el mayor número posible de correcciones y mejoras previo a cada compilación, esto es, aumentar el inventario.

Una solución alternativa es mejorar las herramientas y procedimientos utilizados: el uso de compiladores incrementales capaces detectar las dependencias y actualizar únicamente aquellos objetos modificados desde la última construcción reducirá significativamente el coste de este proceso. El uso de esta técnica en la actualidad está bastante extendido (Maven es un estándar de facto en la industria del desarrollo Java capaz de gestionar estas dependencias).

Es posible realizar una reingeniería de procesos y analizar el flujo de trabajo para encontrar alternativas más rápidas y económicas. Para esta cuestión puede utilizarse una metodología similar a la expuesta con anterioridad: separar aquellas tareas que pueden han de realizarse online de las que pueden acometerse offline y aplicar técnicas de simplificación y que reduzcan los tiempos para realizarlas.

- **Tareas online**: una reunión con el cliente para establecer los requisitos no puede realizarse offline, si bien es posible realizarla paralelamente con otra tarea online con la que no exista dependencia, para reducir el tiempo de ciclo.
- **Tareas offline**: apunta el autor al desarrollo de parte de la plataforma, por ejemplo los *wirgets* gráficos que, además, suelen ser candidatos de componentes reutilizables.

4.8.6. VSM

El uso de esta herramienta es tratado por Poppendieck con varios ejemplos para entender cómo realizarlas e interpretarlas.
Las cadenas de valor comienzan y acaban con un cliente: desde que éste realiza una petición hasta que se pone a su disposición.
Representan la secuencia de actividades que se llevan a cabo, el promedio de su duración y esperas entre cada una de ellas, identificando el tiempo empleado en dar valor al producto final del que no lo da.

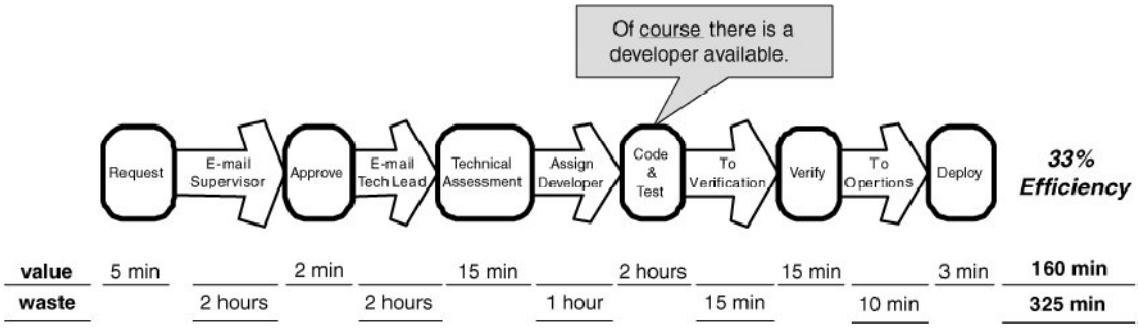

Figura 25 – VSM: Ejemplo de proceso de desarrollo software

Su utilidad, más allá de la representación de la situación actual es el diagnóstico de la misma para localizar focos de mejora: los retrasos y bucles de trabajo rehecho son los primeros candidato.

Los retrasos suelen identificarse con colas de trabajo demasiado largas, falta de capacidad de la organización para acometerlo.

Otros ejemplos de retraso son: excesiva burocracia, problemas con la sincronización de tareas dependientes, falta de involucración de algún agente implicado en el proceso, etc.

Una vez localizados todas los focos de mejora, se deben plasmar en un mapa de la cadena de valor "futuro" que ayude a la implementación de la nueva solución y que

deberá ser actualizado y diagnosticado con el resultado del cambio, siguiendo el principio de mejora continua.

Si observamos el mapeo del flujo de valor de una metodología ágil, podemos comprobar cómo mejora la proporción de tiempo dedicado a dar valor al producto final:

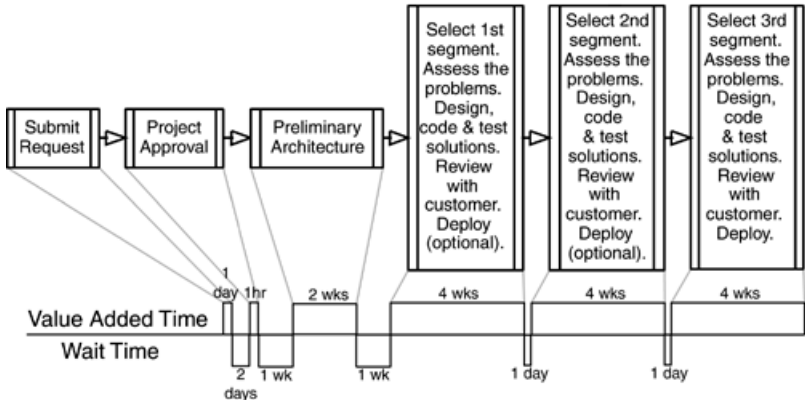

Figura 26 – VSM: Flujo de valor en una aproximación ágil

En otro ejemplo que responde a otra tipología ágil, se pone de manifiesto cómo a menudo las actividades que no son de desarrollo son precisamente funcionan como cuellos de botella y cómo los mayores retrasos se producen tras el desarrollo y pruebas, en actividades y están relacionadas con la obtención de la aprobación del producto final por parte del cliente para disponerlo en el sistema en producción.

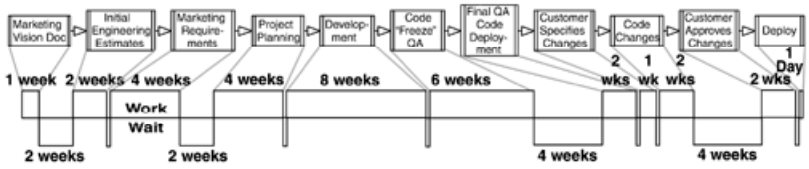

Figura 27 - VSM: la aprobación del cliente como cuello de botella

Se recomienda por el autor que se ha de tratar extender esta técnica a los procesos del cliente: entender cómo crean valor los clientes es una herramienta eficaz para ayudarlos a mejorar.
Otro artículo revisado señala como una de las prioridades para tener competitividad en el negocio de la personalización de software (SPC) es el plazo de entrega. En el mismo, se destaca la aplicabilidad de la metodología VSM para identificar soluciones válidas tanto para la mejora de procesos en general y para los procesos de personalización en particular.

La importancia de la velocidad en este ámbito se debe a varios factores, uno de ellos para reduce el riesgo de quedar fuera del mercado, dando una mayor adaptabilidad del producto ofreciendo a los clientes características personalizadas antes que el resto de los competidores. Otra ventaja es que cuanto antes se disponga de un requisito implementado, más probable es que responda a la necesidad que lo motiva.

En este artículo, se aplica la técnica VSM, según el siguiente diagrama:

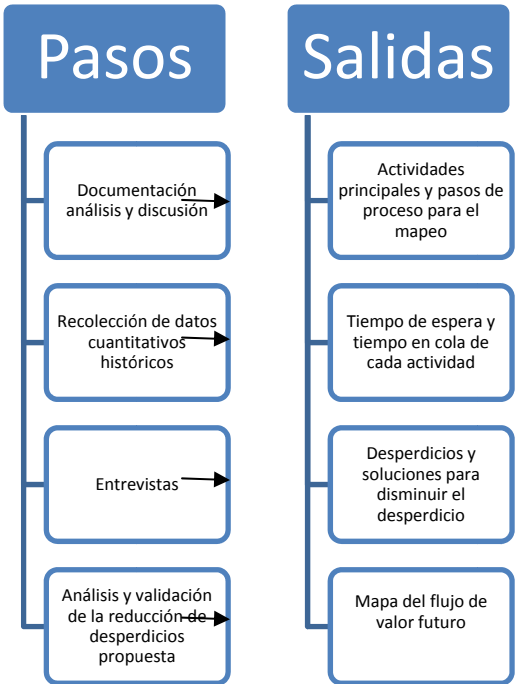

Figura 28 – Aplicación de VSM en I. del Software: pasos y salidas

Los resultados obtenidos de la aplicación fueron de gran utilidad, encontrándose que el 38% del tiempo total dedicado a las actividades de personalización no añadía valor.

Las lecciones aprendidas que señalan son las siguientes:

- **Comprometerse tan tarde como sea posible**: lo que confirma uno de los principios que expone Poppendieck “decidir tan tarde como sea posible”.
- **Desarrollo concurrente**: para reducir el tiempo de ciclo. En particular destacan la importancia de adelantar en lo posible las actividades de pruebas y aprovechar las ventajas de la detección y corrección temprana de errores.
- **Responsabilidad del cliente**: si bien no se encontró una solución para reducir los tiempos de espera de la firma del contrato por parte del cliente, se propone el uso de estrategias que favorezcan la toma de decisiones en este ámbito, como algún tipo de incentivo adicional.
- **Ubicación del equipo de desarrollo:** pone de manifiesto la importancia para el entendimiento de los requisitos que el equipo de desarrollo esté ubicado en un mismo lugar durante todo el proyecto o, al menos, durante un periodo de tiempo.

Así mismo, se pusieron de manifiesto una serie de riesgos asociados al cambio propuesto en el VSM futuro:

- **Pruebas en paralelo**: al desplazar las pruebas en paralelo, disminuye el conjunto de pruebas realizadas a la aplicación finalizada. Se hace necesario mecanismos que incorporen la calidad al proceso y el uso de estándares que minimicen la aparición de problemas.
- **Cuellos de botella por la falta de expertos:** en el mapa futuro se identifica una nueva actividad que requiere de la participación de personal experto, constituyéndose como un cuello de botella potencial. Para mitigar este riesgo se aconseja el uso de técnicas para la transmisión del conocimiento.

4.8.7. Just In Time y SaaS

La filosofía Just In Time tiene una especial importancia en el modelo de negocio tipo SaaS. La disposición de los recursos en función de la demanda en los sistemas de este tipo es fundamental, dado que ésta no es constante, y si bien suele tener picos y valles característicos es habitual que se necesite hacer frente a picos de demanda imprevistos. En este sentido, disponer de tantos recursos como el peor caso posible es prohibitivo y se hace necesario establecer mecanismos que den respuesta rápida a un amento de la demanda.

El concepto de cloud computing o "la nube" se introdujo por empresas como Google o Amazon para dar respuesta a este problema. La nube es conformada por un conjunto de servidores configurados de forma similar y que trabajan al unísono para proveer rápida y eficientemente de recursos hardware a las aplicaciones que se ejecutan sobre ellos.

Este modelo es escalable, de forma que se pueden añadir o eliminar servidores del conjunto modificando la capacidad máxima del mismo. Fuera aparte de la complejidad del software implementado para gestionar estas nubes, el concepto de arquitectura modular sobre el que se basan en su forma más simple puede ser aplicable para cualquier SaaS.

La efectividad de la filosofía Just In Time en este contexto, señala el autor, es mayor conjuntamente con Jidoka, Kaizen y Poka Yokes.
Mientras los dos primeros dan mecanismos para entender y dar respuesta al cambio con rapidez, los poka yoke suponen mecanismos facilitadores de un despliegue de software más frecuente y con menos errores. Todos estos elementos utilizados en una infraestructura modular pueden ser utilizados para, dado un cambio en los patrones del uso del sistema, lanzar de manera automática o guiada inteligente un proceso de reconfiguración del sistema en producción, asignando nuevos recursos.

En cuanto a los recursos "ociosos" en momentos de escasa demanda, pueden ser utilizados para procesos con baja prioridad que no requieran interacción (generación de informes y estadísticas, cálculos, etc.) o para la propia explotación de la organización proveedora del servicio (por ejemplo, para mejorar la capacidad de los entornos de desarrollo y pruebas).

4.9. Jidoka

Los ejemplos más extendidos de la aplicación de Jidoka a la producción de Software tratan sobre la automatización de pruebas y sobre el concepto de integración continua, si bien existen infinidad de ejemplos aplicados para casi la totalidad de los conceptos

tratados con anterioridad, se señala que no se ha explotado completamente este concepto y propone la arquitectura de una herramienta que implemente Jidoka de una manera integral.

Por otra parte, tenemos el concepto de poka-yoke, con una amplia aplicación en la ingeniería del software, tanto para la prevención como para la detección de problemas. Ya en 1993 en "Writing solid code" de Steve Maguire, se presentan una serie de técnicas para resolver dos preguntas: "¿Cómo puedo detectar automáticamente este error?" y "¿Cómo puedo prevenir este error? En cuanto a las referencias en publicaciones que explícitamente utilizan el término poka-yoke, son bastante más escasas.

4.9.1. Herramienta de soporte Jidoka

Se propone una herramienta según el siguiente diagrama:

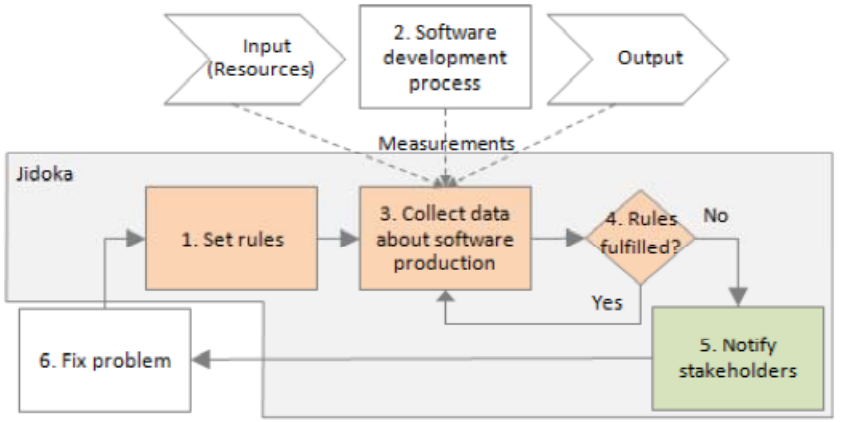

Figura 29 - Herramienta de soporte Jidoka

La arquitectura propuesta contempla:

- Monitorizar además de los **artefactos** que se producen (ya existen herramientas como PMD, ArgoUML, etc. Que monitorizan diversos componentes del Software) los **recursos** que se utilizan para ello y las **actividades** que se realizan, para obtener una visión global del proceso de producción de software.
- Disponer de un sistema que permita definir reglas de calidad, y que de manera integradas en el proceso se detecten automáticamente situaciones críticas, relacionadas con lo anterior, que deben ser corregidas (Esto es posible hacerlo en la actualidad, por ejemplo, mediante el *IDE* Eclipse)
- Mediante el registro automático de información:
    - Del consumo de recursos automatizada, y que contemplando además del tiempo de codificación, otras tareas como son las dedicadas a pruebas o a la redacción de documentación.

    - Del resultado del proceso de desarrollo, de aquellos indicadores necesarios para estimar si un producto es conforme a los requisitos que debe cumplir.
- Interrupción del proceso de producción de software, en función de la violación de las reglas definidas, para evitar mudas, si bien realmente propone un aviso integrado en el IDE que de una forma visual alerte acerca de la regla violada.

4.9.2. Automatización de pruebas

Las pruebas del Software son uno de los elementos donde la automatización aporta mayores beneficios. La automatización, en este contexto, es el uso de estrategias, herramientas y artefactos que reducen en mayor o menor medida la intervención o involucración humana en tareas repetitivas o redundantes y escasa cualificación.

La necesidad de automatización de las pruebas viene dada por la naturaleza sistemática y repetitiva del proceso de pruebas. Además, a medida que los plazos de entrega se acercan y la carga de trabajo aumenta, una de las actividades que se sacrifican son precisamente las pruebas, viendo considerablemente reducido el tiempo que se dedica a ellas, en el mejor de los casos eligiendo un subconjunto de las mismas en base a un análisis de riesgos para determinar cuáles son más importantes.

La automatización de las pruebas es por tanto una estrategia a considerar dada la importancia de no escatimar en la realización de pruebas: cualquier cambio en un sistema puede llevar consigo la introducción de un error (incluso cuando el cambio se realiza precisamente para corregir un error anterior) y es fundamental localizar los problemas de manera temprana. Automatizar este proceso puede ser fundamental para evitar los problemas de la baja calidad.

En este sentido, plantea una serie de ejemplos de métodos aplicados para la automatización de pruebas:

Template (plantilla)	Tests Scripts (Códigos de pruebas)	Images (imágenes)
• Esqueleto de un artefacto que contiene el formato y sirve de guía y punto de partida para la creación de un artefacto • **Ejemplo:** Pantillas de casos de test o de Planes de test (pueden ser creadas por la organización, dispuestas por una herramienta como eclipse con su integración con junit, etc.)	• conjunto de instrucciones codificadas e interpretables por una máquina que automatizan la ejecución de las pruebas • **Ejemplo:** Codificación mediante un framework (p.e. httpunit) de las pruebas de acceso y login a un portal web	• copia de un conjunto de elementos de un sistema (ficheros, bases de datos, etc.) utilizadas para poner rápidamente el sistema en un determinado estado (setup o preparación para realizar pruebas) • **Ejemplo:** Imagen o copia de seguridad de la base de datos con un conjunto de datos conocido

Macros	Batch files (ficheros por lotes)
• conjunto de instrucciones codificadas e interpretables por una máquina (normalmente en el contexto de una determinada aplicación) que permiten la automatización de un conjunto de tareas específicas • **Ejemplo:** Una macro de Excel que capture, de formato y cargue datos para la realización de un informe	• conjunto de instrucciones codificadas e interpretables por una máquina (normalmente en el contexto del sistema operativo o un IDE) que automatizan la ejecución de un conjunto de tareas específicas. • **Ejemplo:** Instrucciones utilizadas para instalar o configurar determinadas opciones del sistema utilizando la consola del SQL-Server

Tabla 3 – Ejemplos automatización de pruebas

D. Haynes invita a llegar más allá de los métodos expuestos, proponiendo al lector a definir qué entiende por automatización y el posterior establecimiento de directrices de automatización para aplicarlas a cualquier actividad candidata a ser automatizada.

En este sentido, propone la aplicación de la siguiente estrategia:

- **Definir dónde encaja la automatización**
    - Localizando las áreas específicas candidatas a automatizar, comenzando por aquellos escenarios más redundantes y eligiendo las áreas más estables de la aplicación sobre las más volátiles.
    - Automatizando aquellas tareas repetitivas más aburridas o susceptibles a provocar errores humanos.
    - Con un enfoque de un buen desarrollo y entendimiento de los escenarios en primer lugar.
    - Utilizar técnicas de pruebas dirigidas por datos (Datadriven testing) para ampliar la cobertura de las pruebas.

- **Planificar más pruebas**
    - Aprovechar el ahorro de la automatización para utilizar otros métodos de prueba: *pruebas exploratorias*, pruebas

  - de configuración.
  - Incrementar el número de pruebas manuales para comprobar los requisitos de alto riesgo.
  - Decidir qué se automatiza
  - Asegurar que si una automatización no puede ejecutase se realice manualmente.

- **Considerar la automatización como una inversión**

  - Formación en el uso de las herramientas automatizadas
  - Asegurar la reutilización de la codificación de los automatismos, documentar para facilitarla.
  - Las pruebas deben ser de fácil mantenimiento: modulares y pequeñas.

- **Implementar la automatización de pruebas de forma iterativa**

  - No pretender automatizar todas las pruebas en un día, aumentar la experiencia en esta técnica e implementar paulatinamente.

Son ventajas de la automatización:

- Software de mayor calidad, más probado, en menos tiempo y con menos recursos
- Potencial para ampliar la cobertura de las pruebas
- Ganar tiempo para profundizar en otras actividades:

  - Planificación de las pruebas
  - Diseño de las pruebas a mayor detalle
  - Pruebas más complejas, más próximas al funcionamiento real de la aplicación y por tanto, más efectivas.

Son ventajas intangibles para las personas que realizan las pruebas:

- La oportunidad de obtener nuevas habilidades (las expuestas para este nuevo enfoque)
- La oportunidad de aprender más sobre el sistema a probar dado que la automatización implica una mayor profundización en el entendimiento del sistema a probar.

4.9.3. Integración continua

Podemos definir la integración continua como:

"Práctica de desarrollo de software donde los miembros de un equipo integran su trabajo con frecuencia, por lo general una integración por persona al día y varias diarias por equipo. Cada integración es verificada y posiblemente realizada por un sistema automatizado de construcción (incluidas las pruebas). De esta manera se pueden detectar errores en la integración lo más rápido posible. Los equipos de desarrollo

encuentran en este enfoque una solución que los lleva a reducir los problemas de integración y permitir el desarrollo de software rápidamente."

- Martin Fowler [46]

El origen del concepto de integración continua es va de la mano con el nacimiento de las metodologías ágiles.

Esta técnica pone en práctica varios de los conceptos tratados anteriormente, así, por ejemplo, se automatizan procesos con objeto de mejorar la productividad y facilitar la detección temprana de errores y se suelen incorporar elementos de gestión visual acerca del resultado de la integración.

En la siguiente figura podemos ver un esquema de funcionamiento de un sistema de integración continua particularizado para un proyecto Web Java.

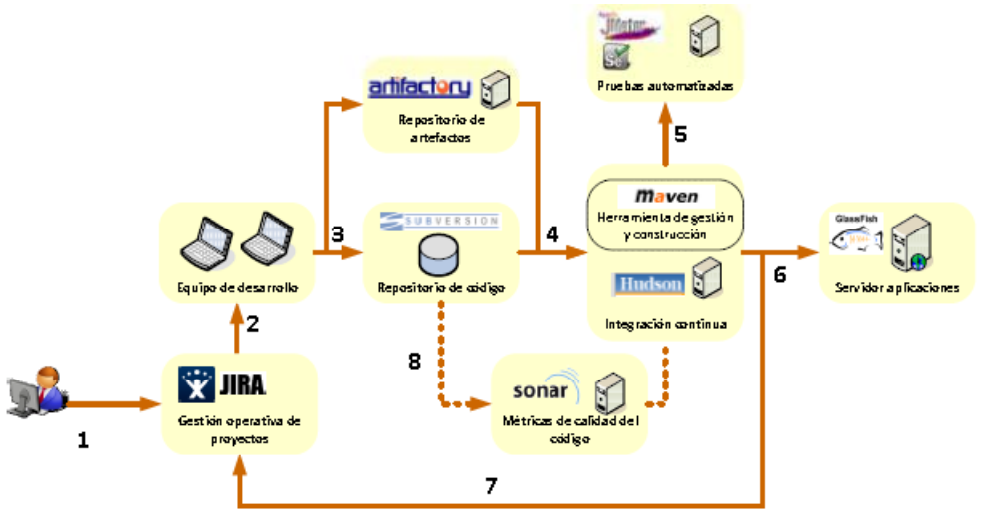

Figura 30 – Sistema de integración continua: funcionamiento

El flujo por el que pasa un requisito hasta su implementación es el siguiente:

1. Se registra el requisito en un sistema de gestión operativa de proyectos, en el ejemplo, JIRA.
2. El equipo de desarrollo recibe la petición y la codifica, así como codifica las pruebas asociadas a la misma.
3. El resultado se envía al repositorio de código (ej. Subversión) y si se realizaron y/o utilizaron artefactos se disponen en el repositorio de artefactos (ej. Artifactory).
4. El sistema de integración continua (ej. Hudson), periódicamente o al detectar un cambio en el repositorio de código, realiza la construcción del sistema. Para esta cuestión hace uso de una herramienta de gestión de la construcción (ej. Maven).
5. Se realizan las pruebas automatizadas (ej. Junit, Selenium, Jmeter)
6. Si se superan las pruebas, es posible desplegar automáticamente el resultado de la construcción en el servidor de aplicaciones.

7. El resultado de la ejecución de la construcción es posible integrarlo en la aplicación de gestión operativa, disponiendo e la misma el conjunto de errores detectados para su solución por el equipo de trabajo, cerrando por tanto, el ciclo de automatización.
8. Es posible incorporar la automatización de métricas de control de calidad del código (Ej. Sonar). La ejecución de este proceso puede hacerse de manera independiente en cualquier momento, aunque típicamente se programa para que se realice nocturnamente.

En el concepto de integración continua, se ven representados en mayor o menor medida algunos de los conceptos Lean:

- **Paradas automáticas**
  Los sistemas IC se detienen en el momento que se detecta un error en la construcción del Software.
- **Separación hombre-máquina**
  Mediante la automatización del proceso de construcción y pruebas.
- **A prueba de errores**
  Mediante la construcción de pruebas específicas para detectar posibles errores.
- **Resolución de la causa raíz de los problemas**
  Un sistema de estas características no permitirá el avance hasta que se corrija la situación que provoca el error.
- **5s**
  Los entornos de integración continua ayudan a disponer, en la práctica, de un *gemba* limpio, ordenado y normalizado.

4.9.4. Resolución de la causa raíz de los problemas

Poppendieck propone el uso del método científico para la resolución de problemas:

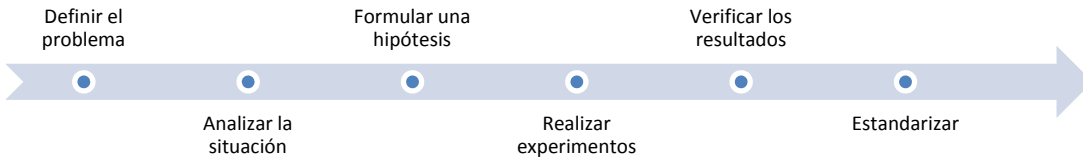

Figura 31 - Método científico para la resolución de problemas

Además, recomienda realizar reuniones para identificar los problemas más importantes y cómo se va a actuar al respecto, así como invitar a la reflexión acerca de los problemas más comunes y cómo hacer las cosas mejor.

4.9.5. A prueba de errores (Poka-yoke)

La propia evolución de los lenguajes de programación a alto nivel pueden considerarse mecanismos de prevención de los errores de una codificación a más bajo nivel.

En cuanto a detección, menciona las herramientas de análisis estático del código que alertan al programador de la necesidad de corregir un error en el momento que se está produciendo. Podemos ver en el siguiente ejemplo como el entorno de desarrollo Eclipse subraya un trozo de código incorrecto y sugiere una serie de acciones que podrían corregir la situación.

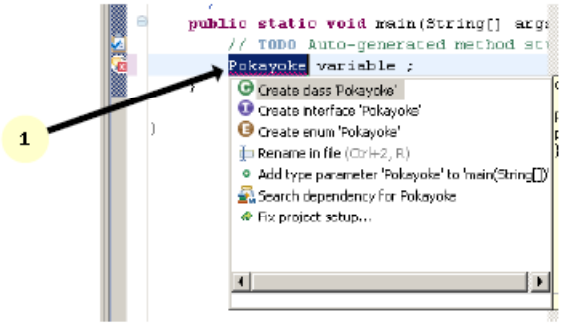

Figura 32 - Poka-yoke en entorno de desarrollo

Por último, destacar una serie de recomendaciones para la creación de poka-yokes en Ingeniería del Software:

- **Simplicidad**: mejor varios sencillos que uno más complejo que cubra toda la casuística.
- **Especificidad**: diseñados para un error en particular.
- **Anticiparse**: maximizar el beneficio que suponen disponiendo de ellos desde el primer momento.
- **Responsabilidad**: corregir los errores en el momento que se detectan.
- **Reutilización**: un poka-yoke efectivo puede ser un modelo a reutilizar.

La importancia de estos mecanismos es aún más crítica en Internet en general y para el modelo de Software como Servicio (SaaS) en particular, donde intervienen simultáneamente cientos de usuarios y, además de la calidad, entra en juego la disponibilidad.

En este escenario, la primera dificultad para solucionar un problema notificado suele ser su reproducción: el comportamiento del sistema en los distintos entornos (desarrollo, pruebas, producción) puede variar de forma inapreciable e irreproducible.

Además, en las metodologías tradicionales suelen existir grandes evolutivos del sistema que al incorporar un número elevado de cambios dificultan la detección de cualquier problema tras la implantación del cambio y a menudo hacen que una vuelta atrás sea prohibitiva.

Las siguientes prácticas y herramientas, se utilizan para combatir estos efectos:

Herramientas de instalación automatizada	Herramientas de automatización en el proceso de contrucción	Bases de datos "activos"
•Para evitar los errores humanos por olvido en el proceso de instalación del software, como Solaris jumpstart o Linux kickstart	•de manera anotada, reproducible y que soporte distintas versiones (para cada entorno), como Hudson o Cruise control	•utilizadas para guardar registro del hardware sobre el que se despliegan los sistemas

Herramientas de monitorización	Empaquetamiento atómico
•que permitan detectar problemas en la infraestructura (caídas, cuellos de botella, etc.)	•disponer el software en paquetes con versiones, autocontenidos e instalables de manera independiente y que permitan ser desinstalados de forma limpia y sencilla

Tabla 4 – Herramientas para combatir los defectos

El uso con éxito de todas estas técnicas y herramientas de manera conjunto permite a empresas como Google o Yahoo:

1. El despliegue continuo de mejoras en sus productos.
2. Reducir drásticamente la cantidad de personal dedicado al control de estas infraestructuras.

4.9.6. Automatismos que ayudan a localizar errores

Es posible hacer uso de una serie de automatismos que, si bien no evitan errores, ayudan a localizarlos cuando estos se producen.

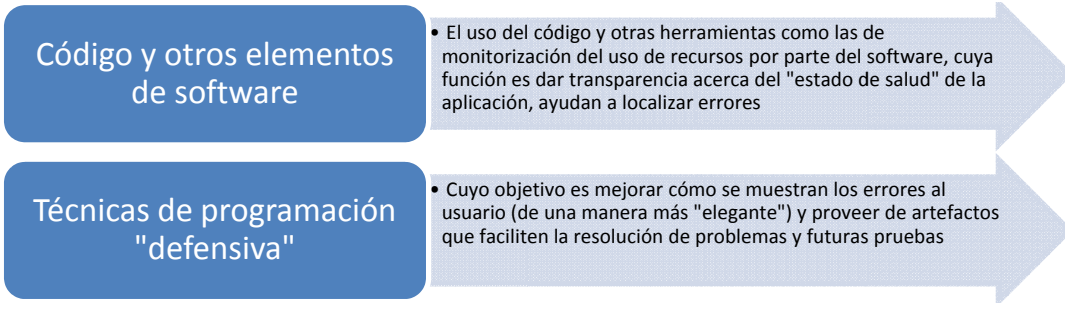

Figura 33 – Automatismos

4.9.7. 5s

El entorno de trabajo en el contexto de la Ingeniería del Software no se limita al espacio físico, sino que abarca un espacio lógico: el escritorio en la pantalla del ordenador, el código base sobre el que trabaja el equipo de trabajo, etc.
Así, una posible interpretación del concepto de 5s para esta disciplina es la siguiente:

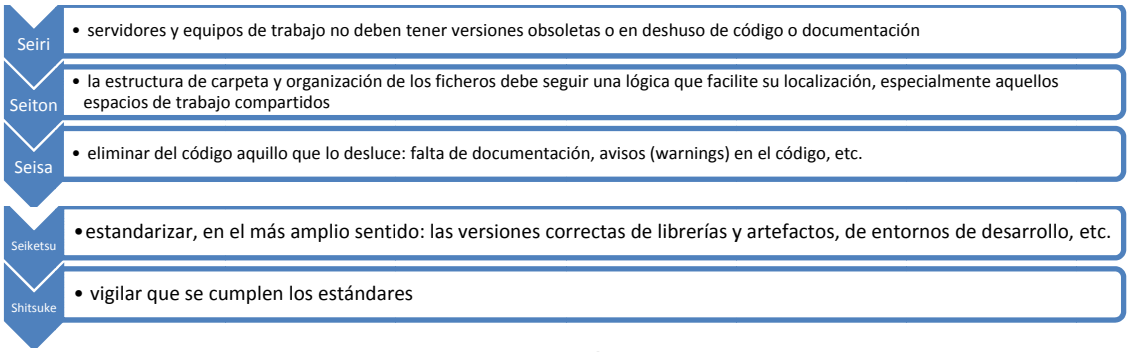

Figura 34 - 5s en Ingeniería del software

Y particularizando para Java, destacar la adaptación de 3 de estos conceptos:

1. **Seiri**: eliminar importaciones, variables, métodos y clases sin utilizar. Refactorizar el código redundante.
2. **Seiton**: minimizar las dependencias, y resolver las dependencias cíclicas entre paquetes.
3. **Seiso**: resolver los fallos detectados en las pruebas unitarias y mejorar la cobertura de las mismas (>80%) y su rendimiento (y el rendimiento del resto de pruebas). Solucionar problemas de estilo de codificación (PMD, Checkstyle). Realizar (o eliminar) los “TODO” (“por hacer”).

5. CASO PRÁCTICO APLICACIÓN METODOLOGÍA LEAN

En octubre de 2008 fui contratada como en una consultora española y desde el momento de mi incorporación hasta la actualidad he estado trabajando como personal de un departamento de Diseño y Desarrollo (DyD) directamente en el cliente.

Mi cliente (de ahora en adelante Banco / Organización), como ya comenté antes, se trata de uno de los principales bancos españoles y con gran presencia internacional y ha optado por aplicar la metodología Lean, no sólo en el ámbito del desarrollo del software, sino en toda su gestión empresarial.

En este apartado, intentaré reflejar, desde mi experiencia profesional, el ciclo de vida de un desarrollo software, desde que en el cliente surge una necesidad, hasta su implantación en el entorno de producción aplicando la metodología Lean que exige el banco.

5.1. Descripción de la organización

El banco, es un grupo financiero global presente en más de 31 países a lo largo del mundo, cuenta con más de 114.000 empleados y da servicio a 51 millones de clientes y con un modelo de negocio minorista centrado en el cliente, que ofrece a sus clientes en todo el mundo una gama completa de productos y servicios financieros y no financieros

Disfruta de una sólida posición de liderazgo en el mercado español, es la mayor institución financiera de México y cuenta con franquicias líderes en América del Sur y en la región “Sunbelt" de EE. UU. Además, cuenta con una presencia relevante en la banca de Turquía (a través de inversiones estratégicas en Garanti Bank), y opera en una amplia red de oficinas en todo el mundo.

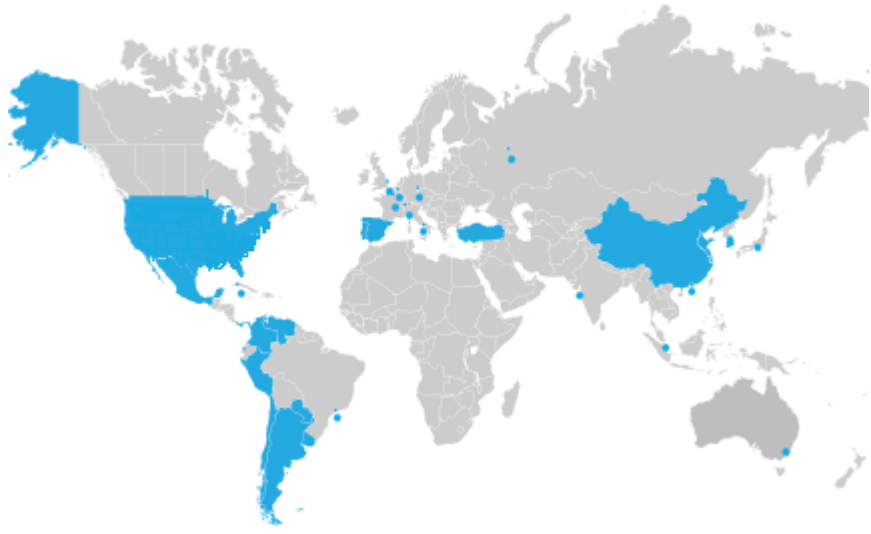

Figura 35. Zonas geográficas en las que tiene presencia el banco

Datos básicos

- 114.228 empleados
- 51 millones de clientes
- Presente en 31 países
- 8.135 oficinas

La visión del Grupo se define como “trabajar por un futuro mejor para las personas”, mientras que **el objetivo general es establecer relaciones duraderas con los clientes**.

Cuenta con una sólida cultura corporativa que define la vida del Grupo, que incide en su manera de actuar y que le permite afrontar con éxito los retos del futuro.
Los empleados, en sus relaciones diarias con los clientes, son la voz de la marca. Más allá de las relaciones comerciales, de la actividad en el sector financiero, se tiene claro que se trabaja por un futuro mejor para las personas.

Para desarrollar esa visión se han establecido siete principios corporativos, que se materializan en compromisos con los clientes, con los empleados, con los accionistas y con la sociedad en general, y se concretan en criterios operativos.

Los siete principios corporativos son:

1. El cliente como centro de nuestro negocio.
2. La creación de valor para nuestros accionistas como resultado de nuestra actividad.
3. El equipo como artífice de la generación de valor.
4. El estilo de gestión como generador de entusiasmo.
5. El comportamiento ético e integridad personal y profesional como forma de entender y desarrollar nuestra actividad.
6. La innovación como palanca de progreso.
7. La responsabilidad social corporativa como compromiso con el desarrollo.

Me parece muy importante presentar también lo que definen como “Marca y Reputación” y que explica el por qué de la implantación de la filosofía Lean en su modelo de negocio.

Marca y Reputación

“Trabajamos por un futuro mejor para las personas”

Esta visión será siempre relevante y es una guía permanente en todo lo que se hace, pero el significado que las personas dan a un futuro mejor evoluciona a lo largo del tiempo.

El posicionamiento de la marca tiene que recoger, en cada momento, el significado de la visión más relevante para la sociedad y los clientes con el fin de construir una marca al servicio del negocio.

¿Qué sabemos ahora? Que la sociedad exige cada vez más de los bancos el reconocimiento del poder del cliente. Que los clientes buscan relaciones horizontales y un trato cercano por parte de los asesores financieros. Buscan seguridad, integridad, transparencia y que les hablen en un leguaje fácil que les ayude a la toma informada de decisiones. Además, el tiempo se está convirtiendo en un recurso muy escaso y las personas quieren acceder a los servicios financieros en cualquier lugar y momento.

Por eso, lo que queremos que los demás piensen de nosotros se basa en dos valores fundamentales: ser una entidad confiable y sencilla. Y de estos dos elementos la sencillez es el que cuenta con más potencial para diferenciarnos.

Nos gustaría que el cliente piense que: es el banco que más le beneficia porque le hace la vida más fácil.

Sencillez o vida más fácil significa cuatro cosas:

- Agilidad y pocos trámites
- Conveniencia y multicanalidad
- Cercanía y sencillez en el trato (versus arrogancia)
- Lenguaje claro, transparente y fácil de entender

El posicionamiento tiene que actuar como filtro en todo lo que hacemos y requiere una cultura y unos comportamientos específicos por parte de los empleados.

Se entiende la marca como una 'marca experiencia', es decir, que la marca no es sólo la promesa que hacemos, sino también las experiencias que nuestros empleados entregan a todos los que se relacionan con nosotros. Por eso, Marca y Cultura no se pueden entender de manera separada.

Una marca fuerte, una marca con buena reputación, requiere precisamente una alineación entre lo que decimos y lo que hacemos que además sea coherente a lo largo del tiempo. La gestión de la reputación es un elemento clave y una palanca de diferenciación sostenible.

En 2010, el grupo lanzó una nueva identidad corporativa que apuesta por:

1. Un estilo de comunicación sencillo y diferencial, alineado con el posicionamiento de la marca.
2. El uso de un lenguaje claro y transparente.
3. Una identidad visual única que permita reconocer la marca rápidamente en todas sus aplicaciones y canales.

Como vemos, aparece varias veces el concepto de sencillez y evolución para adaptarse a las nuevas situaciones, claves en la metodología Lean.

Organigrama

En la siguiente figura podemos ver parte del organigrama del banco. Está detallado hasta el departamento Sistemas Globales de Auditoría, al cual pertenezco.

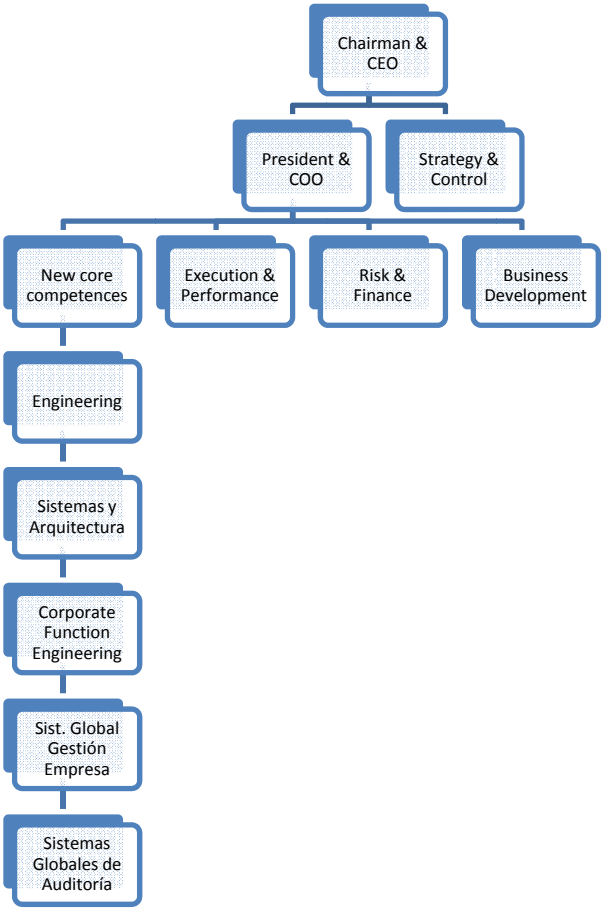

Figura 36. Parte del organigrama del banco

Fundamentos de la gestión LEAN en la organización

La “gestión lean” es una filosofía de trabajo cuyo nacimiento tiene lugar en el ámbito de la empresa Toyota, impulsada por su ingeniero Taiichi Ohno. Esta persona, que llegó a vicepresidente de la compañía, implantó a lo largo de su carrera un cambio de punto de vista de la producción, entendiéndola como un proceso que se ve desde atrás (pull), en el que los productos finales van tirando de los productos semiterminados y de las materias primas; en lugar del enfoque tradicional (push), que partía de las materias primas y avanzaba hacia el producto terminado, el cual generaba múltiples ineficiencias. Todo ello impulsó esta nueva filosofía de hacer las cosas.

El término “lean” viene del inglés y significa “delgado, flaco, esbelto”. En el contexto de la empresa, el significado que mejor encaja es “ajustado”. La gestión lean es una filosofía de trabajo que busca a toda cosa la eliminación del despilfarro y la aportación de valor al cliente, en todos los procesos de la organización y que se basa en los siguientes tres pilares fundamentales:

Aportar valor al cliente: Aportar valor real al cliente, dándole un producto o servicio que resuelva su problema completamente, minimizando el coste, proporcionando exactamente aquello requiere, dónde y cuándo lo demande.

Eliminar el despilfarro: Lucha sin tregua para detectar y eliminar el despilfarro (o muda); entendiendo el despilfarro como todas las actividades de un proceso que no aportan valor al cliente y que sí añaden coste. Todas las tareas que se ejecutan y no aprecia el cliente, sobran.

Desarrollar procesos flexibles: Que sean capaces de adaptarse rápidamente la demanda y de satisfacer las necesidades del cliente en cualquier momento, en términos de tipo de producto (diseño) y de capacidad (escalabilidad).

A partir de los 3 pilares básicos, la gestión lean propone los siguientes seis principios fundamentales:

Resolver el problema del consumidor completamente.

Minimizar el coste del consumidor y también del coste del proveedor.

Proporcionar exactamente aquello que se requiere.

Entregar el valor donde se requiera.

Ofrecer el valor cuando se requiera.

Proporcionar el valor que realmente se desea, no solo las opciones existentes.

Cuando se leen los seis principios anteriores, se puede comprobar que la gestión lean es una filosofía de trabajo muy sencilla, que persigue unos objetivos claros, de aportar valor al cliente al mínimo coste, a través de la reducción del despilfarro. Pura lógica, la de crear productos o servicios que resuelvan los problemas completos de los clientes, al mínimo coste, donde y cuando lo requiera el cliente. Entonces ¿Por qué hoy en día hay tantas empresas que en la práctica están alejadas de esta visión de la gestión lean?

Diferencias entre la gestión lean y la gestión convencional

Las siguientes, son algunas de las diferencias más importantes entre la gestión lean y la gestión convencional, que se pueden apreciar en muchas empresas. No siempre ocurren todas, pero es bastante común encontrarse con algunas como, por ejemplo:

La gestión lean promueve los sistemas pull, en los que la demanda real activa la producción; mientras que la gestión convencional promueve los sistemas push, es decir, que empujan la demanda al mercado.

La filosofía lean busca la excelencia y la perfección, observando y escuchando directamente al cliente; mientras que la gestión convencional se mueve a través de

prácticas de vigilancia a la competencia, con herramientas como el benchmarking o los estudios de mercado tradicionales, por ejemplo.

La gestión lean supone trabajar con una oferta constantemente ajustada al mercado, con un stock que idealmente es igual a cero; mientras que la gestión convencional necesita las ofertas y descuentos para colocar sus tradicionalmente altos niveles de stock en el mercado.

La gestión convencional busca resultados a corto plazo, en una única etapa; mientras que la gestión lean persigue los resultados a medio y largo plazo, en varias etapas, a través de la mejora continua.

La gestión lean promueve una visión global del negocio, como sistema completo; mientras que la gestión convencional promueve una visión más local, es decir, por departamentos a modo de islas.

Conclusiones

En definitiva, la gestión lean promueve hacer las cosas con un foco apuntando hacia el cliente, buscando darle lo que quiere, cuando lo quiere y cómo lo quiere, al mínimo coste, siempre buscando la perfección y con una visión de sistema. Para una empresa que trabaja según la filosofía de gestión lean, dar dos pasos adelante y uno hacia atrás es aceptable; no dar pasos adelante, no lo es. Es el “simplemente hágalo”, el hecho de que cuando se acaba de conseguir que algo funcione bien, ha llegado el momento de mejorarlo de nuevo.

La gestión convencional, busca más satisfacer las necesidades inmediatas de los accionistas, a través de una mentalidad económico-financiera que se ha puesto por encima de la realidad de crear valor para el consumidor. No baja al taller y se ensucia las manos, sino que permite que la lo que suceda en él no tenga nada que ver con lo que pasa en realidad, es decir, lo que dice el mercado. Gestiona operaciones aisladas, en lugar de procesos, sin establecer flujos de tareas de forma encadenada. Es una forma de supervivencia pura y dura.

5.2. Necesidad surgida

Podemos decir que la crisis económica ha provocado cambios y las empresas han tomado decisiones en muchos ámbitos de la gestión: financieras, de producto o servicio, de recursos, a menudo con implicaciones organizativas.
En la coyuntura de estos últimos años el ámbito del cambio es el económico (principalmente) y la inestabilidad, una de las más grandes de la historia. Es en este momento de gran cambio que estamos sufriendo en el que las empresas han de adecuarse a los nuevos tiempos que vienen. A este fenómeno se le llama evolucionar. Esta organización ha elegido adoptar la filosofía Lean para llevar a cabo su evolución.
Como hemos visto en la descripción de la organización, es en 2010 cuando se define una nueva identidad corporativa. La organización pretende que se la reconozca por hacer la vida más fácil al cliente, adaptándose a todas sus necesidades de una forma clara y sencilla.
Muchas organizaciones están llevando a cabo su transformación siguiendo los principios y fundamentos de Lean, de hecho, recientemente se ha puesto en marcha la Asociación Española de Lean IT (AELIT) con la misión de promover las mejores prácticas, -en el

ámbito de las TIC-, para la búsqueda de la eficiencia, la calidad, el rendimiento, la innovación y la mejora continua en las organizaciones, tomando como pilar clave a las personas y empleando técnicas y herramientas de Lean.
Así, esa asociación nace con vocación de transmitir los principios Lean y vertebrar su actividad a través de grupos y foros de discusión donde se compartirán experiencias reales de adopción de Lean IT y la organización de sesiones formativas y talleres Lean donde se darán a conocer las principales técnicas, herramientas y principios de Lean IT. La asociación considera que la aplicación de los principios Lean ha demostrado una eficacia incuestionable en el incremento de la eficiencia y productividad, mediante la involucración directa de los empleados en la consecución de los objetivos empresariales.
En esta asociación se dan cita empresas de la talla de Quint, Cepsa, Mapfre, Repsol, Grupo Leche Pascual y como no, la del grupo el que hablamos en este proyecto.
A diferencia de otras metodologías, el lean afecta a cada aspecto del trabajo y a todos los trabajadores. Cuando un empleado, del rango que sea, se incorpora a una empresa lean proveniente de una empresa tradicional se da cuenta de que allí sucede algo diferente.
Y si la organización ha optado por seguir la filosofía Lean como un proceso de mejora continua dentro de la empresa, esta misma filosofía se ha tenido que comenzar a aplicar en los departamentos implicados en el desarrollo del software.

5.3. Desarrollo de un proyecto software

Desde que en 2008 comencé trabajando para esta organización como programador junior, hasta el día de hoy en el que sigo trabajando como Analista Orgánico, siempre hemos desarrollado evolutivos sobre aplicaciones que pertenecen al departamento de Auditoría Interna.

En concreto gestionamos las siguientes aplicaciones:

- **Aplicación de auditoría interna**: Se trata de una aplicación con más de 15 años a sus espaldas desde la que poder realizar auditorías a oficinas y/o clientes del banco.

  Esta aplicación tiene varias distribuciones:
  - España y Portugal
  - México
  - Chile
  - Venezuela
  - Colombia
  - Argentina

  Desde sus inicios hasta hoy evoluciona constantemente para adaptarse a la normativa legal vigente en cuanto a análisis de riesgos, normativas del banco de España, normativas locales, etc.

- **Aplicación de seguimiento de recomendaciones**: Se trata de una aplicación desde la que poder realizar auditorías tanto internas a unidades organizativas del propio banco como externas a diversas organizaciones con las que se tiene alguna relación mediante el seguimiento de recomendaciones.

- **Aplicación de gestión de los auditores**: Se trata de una aplicación desde la que gestionar el trabajo de los recursos del departamento de auditoría interna, que trabajan con las dos aplicaciones anteriores. En ella se controla su productividad mediante los trabajos a los que está asignado así como las horas que dedica a cada uno, los gastos imputables a cada trabajo, etc.

- **Nueva aplicación de auditoría interna**: Actualmente se está desarrollando una nueva aplicación para ir sustituyendo progresivamente a la actual de auditoría interna para así tener una única distribución de la aplicación y para ofrecer una interfaz más cómoda, intuitiva y moderna, ya que la actual, con sus más de 15 años se va quedando algo obsoleta en ese sentido.

5.3.1. Departamentos implicados en el desarrollo

Para poder seguir los siguientes apartados voy a describir los departamentos con los que tiene relación el nuestro de Diseño y Desarrollo (DyD)

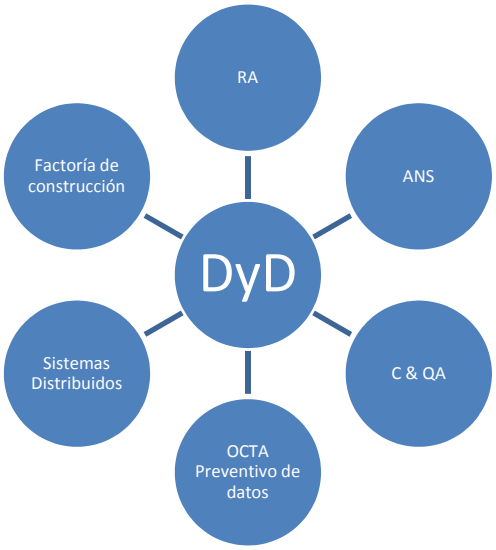

Figura 37. Diagrama departamentos involucrados con DyD en el diseño de software en el banco

DyD: Diseño y Desarrollo. Departamento, en el que me incluyo, y que forma parte de la organización Sistemas Globales de Auditoría. Somos los encargados del desarrollo de los nuevos requerimientos de los usuarios.

Nuestro departamento está formado por:

- un responsable del banco que hará las funciones de interlocutor con el cliente más la mayoría de las relacionadas con la gestión del proyecto
- conjunto de personas externas al banco que formamos el equipo de proyectos

Durante nuestro trabajo tenemos que relacionarlos con los departamentos siguientes.

C&QA: Departamento que sigue que cumplamos la normativa para certificar la calidad del Ciclo de Vida Productivo

OCTA Preventivo de datos: Departamento encargado de la gestión de las bases de datos. Es el encargado de gestionar las modificaciones que sean necesarias en el modelo de datos, asegurando que estas cumplen con la normativa exigida por el banco en cuanto a nomenclatura y más adelante, proporcionándonos un apoyo en caso de cualquier duda sobre cómo obtener información de calidad de la base de datos.

Sistemas Distribuidos: Departamento encargado de la gestión de las máquinas de los entornos previos. Actualmente disponemos de los siguientes entornos:

- Desarrollo
- Integrado
- Aceptación de usuarios
- Producción

Este departamento se encarga de peticiones sobre los 3 primeros. Estas peticiones pueden ser del tipo de creación de directorios, modificación de ficheros de configuración (classpath, fichero de despliegue, ficheros de propiedades), gestión de los servidores de aplicaciones (parada, arranque, reinicios), ejecución de scripts, etc.

RA: Recepción de aplicaciones. Departamento similar al anterior pero exclusivo del entorno de producción. Es el departamento responsable de las implantaciones en el entorno de producción siguiendo las directrices del equipo de DyD, y de cualquier acción sobre los servidores de producción.

ANS: Acuerdo Nivel Servicio. Departamento responsable de la aplicación en el entorno de producción una vez finalizado el proceso de garantía.

Durante el desarrollo de un nuevo evolutivo, el departamento de ANS es informado por el departamento de DyD de todas las fases por las que pasa el desarrollo. De hecho, este departamento debe dar su aprobación para la implantación del desarrollo formalizando una relación contractual por la cual pasará a sus manos la gestión en producción del desarrollo 1 mes después que el usuario comience a usarla. Este período de un mes es lo que conocemos como el proceso de garantía. Durante ese proceso, cualquier incidencia sobre el desarrollo debe ser resuelta por el equipo de DyD. Los términos son negociables en función de la complejidad del desarrollo.

Factoría de construcción: Generalmente la construcción de los componentes diseñados en el desarrollo de la solución software se externaliza en las llamadas Factorías de construcción.

Esto se traduce en que el equipo de proyectos analiza los requerimientos y diseña los componentes necesarios, pero que la construcción se subcontrata y la realiza una factoría externa

5.3.2. Herramientas usadas en el desarrollo:

Las mostramos en distintos apartados, separando las distintas herramientas según su finalidad en el proceso de desarrollo.

- **Herramientas de diseño**

Este tipo de herramientas, como su nombre indica, nacen con el objetivo de facilitar y dar unas pautas a la hora de diseñar piezas de software. En esta línea, se proponen distintos esquemas o tablas para documentar y planificar la estructura de la aplicación a desarrollar. Actualmente, la principal herramienta de diseño es UML, por lo cual este apartado está dedicado de forma exclusiva a este lenguaje y a aplicaciones que nos permiten trabajar con él.

De forma resumida, UML (siglas inglesas de Unified Modeling Language, o lenguaje unificado de modelado) proporciona mecanismos estándar para visualizar, especificar, diseñar y documentar sistemas de software. En términos generales, UML nos permite dibujar el "plano" de una aplicación, es decir, su modelo, incluyendo aspectos generales tales como procesos de la lógica de negocio, funciones, componentes, o incluso otros más concretos como podrían ser diseños de bases de datos o expresiones propias de un lenguaje en particular.

Principalmente, UML propone distintos tipos de diagrama que pretenden plasmar cada aspecto de la aplicación que se está diseñando. En concreto existen 11 tipos de diagrama distintos. En el anexo A se puede encontrar un listado de los tipos de diagrama, con la descripción y un pequeño ejemplo de cada uno.

Por otro lado, han surgido una serie de aplicaciones que pretenden facilitar todo el proceso de diseño mediante UML: los entornos de diseño. Este tipo de aplicaciones permiten crear y editar esquemas UML de forma muy sencilla y cómoda. Habitualmente, se componen de un editor gráfico (similar a cualquier programa de dibujo o edición fotográfica) que incluye distintas formas que se corresponden con los distintos objetos que se pueden encontrar en los diagramas UML. Aparte de esto, también suelen incorporar herramientas adicionales, como por ejemplo generadores automáticos de código (los cuales generan el esqueleto del código según nuestro diseño), ingeniería inversa (para dibujar diseños a partir del código de alguna aplicación), etc.

En los desarrollos para las aplicaciones anteriores se usa el editor UML del entorno de desarrollo integrado RAD (IBM Rational Application Developer), creado por la división Rational Software de IBM para el diseño visual, construcción, pruebas y despliegue de servicios web, portales y aplicaciones JEE.

Este entorno de desarrollo contiene un editor muy completo de UML y dispone de múltiples características, entre ellas lo más destacable es la generación automática de código, el hecho de que permite realizar todos los diagramas UML existentes en la especificación 2.0 y la posibilidad de usarlo conjuntamente con un repositorio, Clear Case en este caso.

En la siguiente imagen se puede ver la interfaz del editor de UML que contiene RAD.

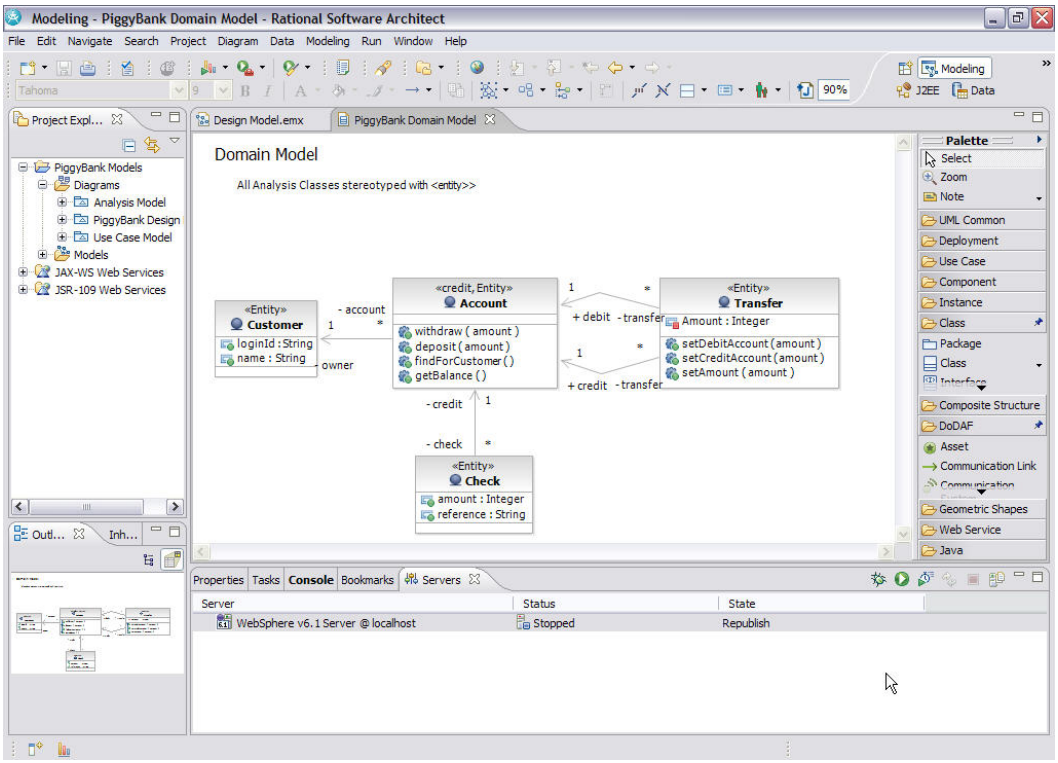

Figura 38. Interfaz gráfica editor UML de RAD

- **Herramientas de programación**

Este tipo de herramientas tienen como finalidad facilitar a los desarrolladores a escribir y testear el código fuente de que se compone la aplicación. Entre las herramientas de este tipo se incluyen entornos de desarrollo (IDE), compiladores o herramientas de mecanización de procesos, como pueden ser “Ant” [5] o el más reciente “maven” [6]. En este apartado se describe la función de cada tipo de herramienta de programación, además de comentar y comparar distintos ejemplos.

IDE

Las siglas inglesas IDE significan Integrated Development Eviroment, es decir, entorno de desarrollo integrado. Este tipo de aplicaciones nos proporcionan ciertas facilidades a la hora de escribir código. Cada IDE tiene características distintas, pero las más comunes son el coloreado del código (con la finalidad de identificar visualmente distintas partes del código como, por ejemplo, variables, comentarios, funciones, etc.), integración con compiladores para compilar el código con un sólo clic y función de debugger para facilitar la localización de errores de programación.

Algunos de los IDE’s más famosos son: Microsoft Visual Studio [7] (pensado para C, C++ y C#), Zend [8] (para editar código PHP), netbeans [9] (Java) o eclipse [10] (en principio pensado para Java, pero dispone de plugins para funcionar en multitud de lenguajes de programación).

En los proyectos, como comenté antes, trabajamos con el IDE RAD (IBM Rational Application Developer). Es un entorno de desarrollo integrado creado por la división Rational Software de IBM para el diseño visual, construcción, pruebas y despliegue de servicios web, portales y aplicaciones JEE.

RAD está basado en el IDE Eclipse y soporta sus extensiones. Incluye herramientas y editores para trabajar con:

- Conexiones a bases de datos y SQL.
- HTML.
- Java.
- JavaServer Faces y JavaServer Pages.
- UML.
- Servicios web.
- XML.

Compiladores

En cuanto a compiladores, hay que destacar que cada lenguaje compilado tiene el suyo y poco más hay que decir, simplemente son programas, habitualmente en modo texto, que permiten convertir el código a un lenguaje entendible para la máquina. Por ello, en este apartado, más que explicar los compiladores como herramienta, se describen herramientas que nos facilitan las tareas "mecánicas" (en el sentido de que son siempre iguales) de compilación, empaquetado y despliegue de las aplicaciones. Las herramientas de este tipo más famosas y utilizadas son *Ant* y *Maven*. Las dos aplicaciones están desarrolladas por la *Apache Software Foundation* [11], y se puede decir que la segunda es una evolución de la primera.

Ant

Ant es una herramienta de construcción de software. Nació con el objetivo de superar las limitaciones que tenían las herramientas existentes con anterioridad (make, gnumake, nmake, jam, etc.). La ventaja que introduce respecto a estas es que no depende de las órdenes de la línea de comandos de cada sistema operativo, sino que está basado en archivos de configuración XML y en clases java. Para conseguirlo, ant incluye multitud de funciones que sustituyen a los comandos clásicos de una línea de comandos. Incluye funciones de creación/eliminación de archivos y carpetas, funciones de empaquetamiento de software, etc.

Por todo ello, teniendo en cuenta que tanto java como XML son multiplataforma, se presenta como una herramienta idónea para soluciones independientes del sistema operativo.
Por otro lado, ant también comporta una serie de limitaciones. La primera de ellas es que es una herramienta basada en XML, con toda la complicación que ello supone para nuevos usuarios y en proyectos de gran envergadura, en los cuales los archivos ant se hacen enormes y muy complejos. Otra limitación seria sus reglas de manejo de errores y el hecho de que no tiene persistencia de estado, con lo cual no puede ser usado con confianza para manejar construcciones largas (de uno o más días).

Usamos esta herramienta para crear los archivos diferentes archivos .jar con el código compilado que componen la distribución de la aplicación.

- **Herramientas de gestión de la documentación**

Como su nombre indica, este tipo de herramientas nos ayuda a organizar la documentación generada, de forma que sea accesible para todos los miembros del equipo. En este apartado describiremos las funciones, ventajas y desventajas de tres gestores documentales, cada uno de ellos con un enfoque muy distinto.

Desde la aplicación de la metodología lean, la organización ha adquirido un dominio google. De esta forma, los trabajadores tenemos a nuestra disposición todas las herramientas google: Gmail, Google calendar, Google drive, Google apps, etc.

Gracias a esto y formando parte de las medidas enfocadas a hacer más sencillo en Ciclo de Vida Productivo se ha creado un Repositorio Único de Documentación, de donde toman la información todos aquellos equipos que realizan cualquier tipo de gestión.

Existen dos tipos de gestión de este repositorio, a día de hoy en Drive, diferenciando si se trata de un Proyecto o de un Evolutivo:

<u>Repositorio Único en Proyectos</u>

En el alta de los Proyectos el equipo de Metodología genera una carpeta con la estructura por fases del proyecto para la gestión de toda la documentación.

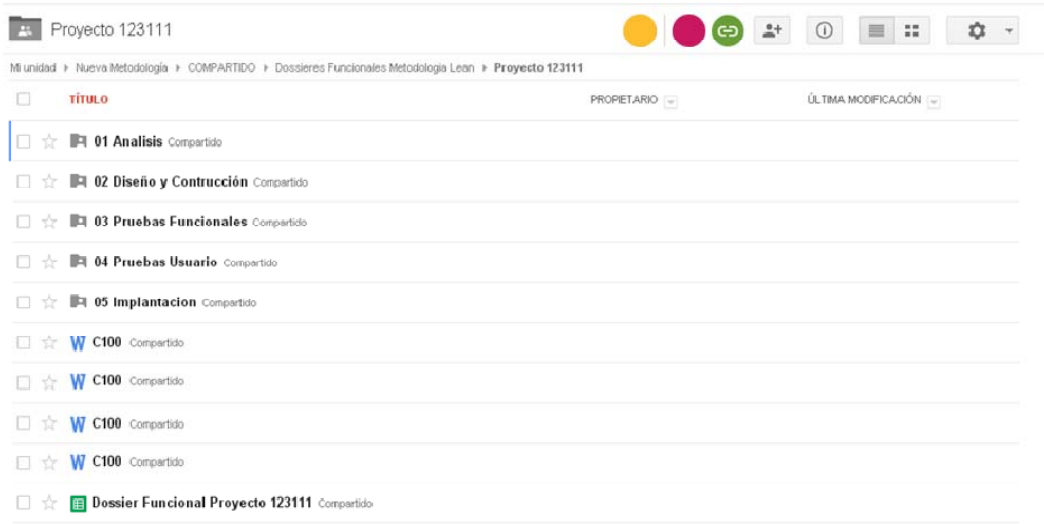

Figura 39. Repositorio Único en Proyectos

Repositorio Único en Evolutivos

En este caso se trata de un trabajo combinado entre el equipo de Metodología y el equipo de Proyecto:

- El equipo de Metodología genera una carpeta por cada UUAA sobre la que se desarrollan evolutivos

- Sobre esta carpeta, con objeto de evitar realizar envíos adicionales de ninguna información por correo electrónico, es necesario que los equipos de Proyecto que no lo estuvieran haciendo hasta ahora:
  - Dentro de dicha carpeta cree una nueva carpeta por cada evolutivo (identificando su nombre con el código de evolutivo del mismo).

  - Incorpore únicamente los siguientes documentos:
    - Documento con el contenido de los casos de las Pruebas Funcionales

    - Documento con la Aprobación de las Pruebas de Usuario

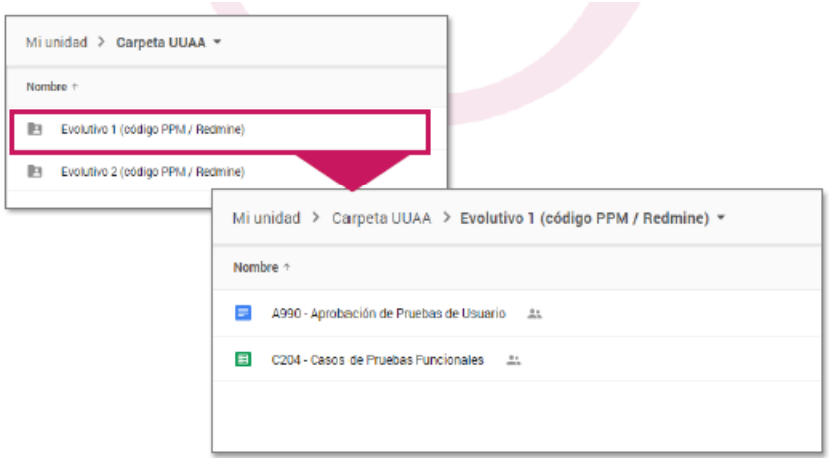

Figura 40. Repositorio Único en Evolutivos

Adicionalmente, queremos se está trabajando en la implantación de una **Nueva Herramienta de Metodología** que actuará como **repositorio único de los documentos**, sustituyendo a la función que realiza actualmente Drive y añadirá otra serie de **capacidades**:

- **Mejoras en las capacidades de búsqueda:** Toda la documentación incorporada podrá ser consultada tanto a nivel Proyecto como a nivel UUAA (nombre de la aplicación)

- **Mejoras en la escalabilidad:** Esta documentación podrá ser evolucionada en fases posteriores

- **Mejoras colaborativas:** Dispondremos de un Repositorio Único de Documentación en un Entorno Colaborativo donde las áreas certificadoras podrán consultar la documentación en el momento en que el equipo de Proyecto indique que está lista, sin necesidad de esperar al cierre de proyecto ni realizar envíos por mail

- **Herramientas de gestión de la documentación**

Este tipo de herramientas ayudan al director del proyecto a planificar los distintos eventos del proyecto, así como asignarlos a los distintos componentes del equipo de desarrollo. Los eventos pueden ser desde la definición de tareas hasta la programación de reuniones o plazos de entrega.

OpenProj

OpenProj es una aplicación de administración de proyectos diseñado como sustituta de sobremesa completo para la suite de ofimática Microsoft Office. En ella principalmente se nos permite hacer la planificación de un proyecto (definición de tareas, asignación de recursos, plazos, etc.), para luego mostrarla con distintas vistas (diagramas de Gantt, de recursos, de relaciones, etc.). Es decir, que más que un programa de gestión de proyectos es un programa de planificación, ya que no implementa flujos de trabajo ni demasiadas opciones a nivel financiero, simplemente permite definir diagramas de planificación. En la siguiente figura se puede ver el aspecto de OpenProj.

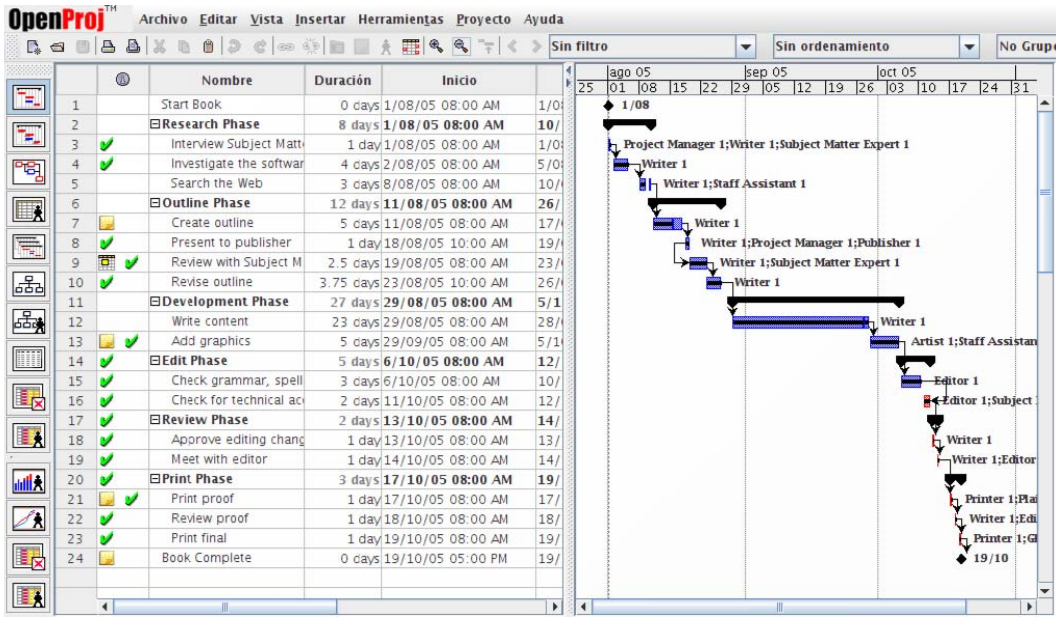

Figura 41. Interfaz gráfica de OpenProj

- **Herramientas de modelado de datos**

Como herramienta para modelar las bases de datos implicadas en nuevos desarrollos siempre hemos usado Power Designer, sin embargo, desde la entrada en la organización de la metodología Lean, ha cambiado el método de trabajar con ella.

PowerDesigner

Es una herramienta para el análisis, diseño inteligente y construcción sólida de una base de datos y un desarrollo orientado a modelos de datos a nivel físico y conceptual.

Con esta herramienta se trabaja directamente sobre los modelos lógicos y físicos de la base de datos a la hora de solicitar modificaciones de base de datos.

Anteriormente, el departamento de diccionario de datos, encargado de revisar que los modelos de datos cumplen con la normativa establecida por el banco, mantenía sus propios modelos físicos.
Los equipos de desarrollo, en diversas unidades de red compartidas, eran los encargados de mantener sus modelos físicos actualizados a medida que éstos iban cambiando.
A la hora de solicitar una modificación era necesario modificar un modelo (que se suponía teníamos actualizado correctamente) e incluirle en una petición Remedy solicitando dicha modificación. Esta petición pasaba por el departamento de diccionario de datos que validaban o no el cambio.

Actualmente, los modelos de datos se encuentran en un único repositorio al que nos conectamos directamente desde la aplicación PowerDesigner. Cuando se necesita una modificación de base de datos, nos conectamos al repositorio y obtenemos una versión en local del modelo. En ella realizamos las modificaciones pertinentes. Las validamos mediante una herramienta de validación de la propia aplicación y las incluimos en una petición Remedy solicitando la modificación.
El departamento encargado de ejecutar la modificación actualizará en el repositorio de PowerDesigner el modelo con las modificaciones incorporadas.

De esta forma, es imposible encontrarnos con un modelo desactualizado.

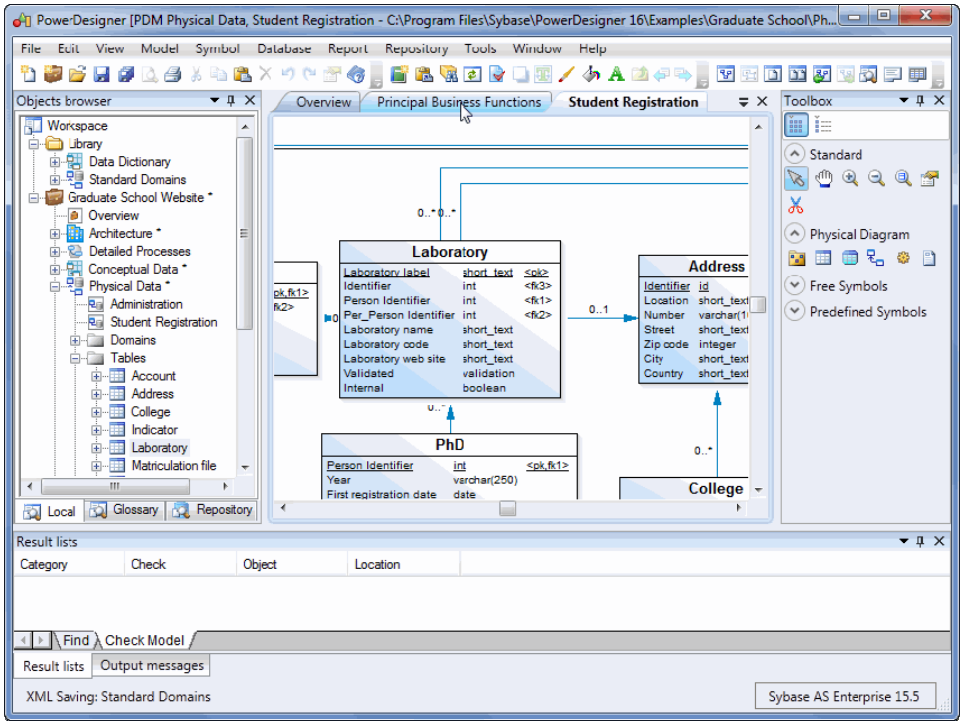

Figura 42. Interfaz gráfica de PowerDesigner

- **Herramientas de trabajo en equipo**

Las herramientas de trabajo en equipo permiten compartir los avances en el proyecto entre los distintos grupos de trabajo que componen el equipo de desarrollo. Las herramientas descritas en los siguientes subapartados (ClearCase y SVN), nos permiten almacenar todo el código fuente desarrollado en un servidor para que todos los componentes del equipo de desarrollo puedan conectarse a este servidor y disponer de todas las partes del proyecto. Estas herramientas son especialmente útiles cuando el trabajo de un componente del equipo depende del de otro.

ClearCase

Rational ClearCase es el sistema de control de versiones de IBM.
Cómo otros sistemas de control de versiones, ClearCase almacena distintas versiones de los elementos que forman parte de un proyecto, que pueden ser directorios o archivos. Estos elementos se almacenan en un repositorio que en ClearCase se llama base de objetos de versionado (VOB).

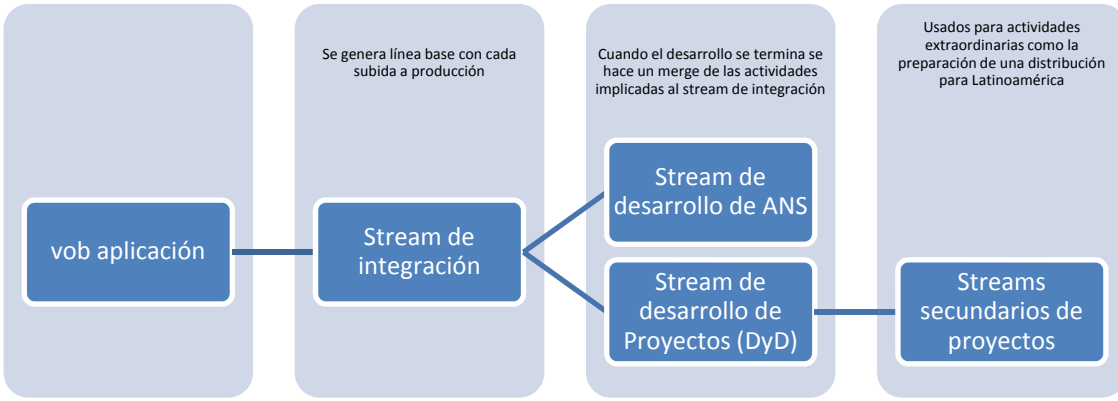

Figura 43. Estructura de los proyectos en ClearCase

Cada miembro del equipo se debe crear una vista sobre el stream de desarrollo correspondiente. En ella realizará las modificaciones de código necesarias. Las modificaciones que se llevan a cabo durante las tareas de desarrollo dentro de la ejecución de un proyecto se registran en actividades, que consisten en varias versiones de distintos elementos de un componente.

Diariamente, cada miembro del equipo libera sus modificaciones para que el resto de miembros, actualizando la vista, tengan el código actualizado. Todos los cambios que un desarrollador registra pueden ser vistos por los demás desarrolladores sin más que hacer un update de su vista, permitiendo que los nuevos cambios se publiquen de manera más inmediata.

Cuando se determina que el desarrollo ha finalizado, se realizará un deliver de las actividades implicadas al stream de integración.

Algo que particularmente considero muy útil a la hora de mantener una aplicación, es que de una forma muy visual obtenemos en forma de árbol el control de las versiones de un componente.

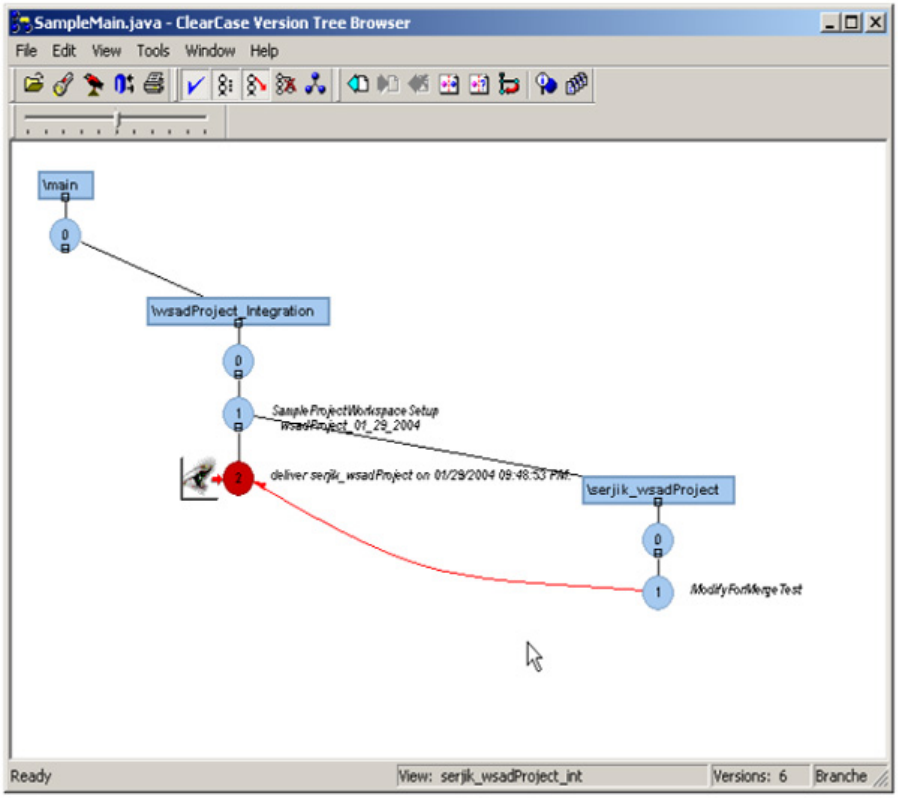

Figura 44. Árbol de versiones de un componente

SVN

Se trata de una herramienta open source de control de versiones cuyo nombre completo es *Subversion*.
Nació con la finalidad de sustituir a otra herramienta open source de control de versiones, CVS y solventar sus limitaciones de diseño. Principalmente funciona de forma similar a CVS, pero introduce una serie de mejoras:

- Se lleva un historial de los archivos y directorios a través de copias y renombrados.
- Se envían sólo las diferencias entre los archivos del servidor y el cliente (en CVS se envían archivos completos).
- Maneja de forma eficiente los archivos binarios (CVS los trata como archivos de texto).
- Permite el bloqueo de archivos.
- Puede ser servido mediante apache, a través de WebDAV.
- Al usarse integrado a apache, puede utilizar todas las opciones que este provee (BBDD, LDAP, PAM, etc.) a la hora de autenticar el acceso a los archivos.

Actualmente, en nuevas aplicaciones se está empezando obliga a usar esta herramienta en vez de ClearCase al tratarse de una herramienta libre.

- **Herramientas de control de calidad**

Las métricas son una serie de pautas objetivas que nos permitirán analizar la calidad del sistema en distintas etapas de su desarrollo. De forma más concreta, las métricas son medidas cuantitativas de algunas propiedades o características de una pieza de software.

Existen multitud de medidas distintas que se pueden tomar sobre una pieza de software. En las siguientes líneas describiremos reglas que se pueden definir consideradas de uso más común:

- Número de líneas de código.
- Complejidad ciclomática.
- Bugs por línea de código.
- Cobertura del código.
- Métrica de punto función.
- Número de líneas de requerimientos.
- Número de clases e interfaces.
- Cohesión
- Dependencia

El banco, tiene definidas un conjunto de reglas. Mediante la herramienta PDM, integrada en el entorno de desarrollo RAD, se escanea y analiza el código para optimizarlo al máximo.

- **Herramientas de gestión de entornos de TI distribuidos**

Para la gestión de las operaciones a realizar en los distintos entornos usamos la suite BMC Remedy IT Service Management Suite.
Desde esta aplicación se solicita a los distintos departamentos de sistemas las operaciones necesarias para el funcionamiento del nuevo desarrollo.
Podemos solicitar distintas acciones. Por ejemplo:

- Operaciones sobre los servidores de aplicaciones (máquinas unix bajo el sistema operativo Solaris)
    - Creación / modificación de directorios
    - Eliminación de ficheros
    - Permisos para usuarios sobre las máquinas
    - Ejecuciones de scripts
- Operaciones sobre la base de datos (DB2 en nuestro caso)
    - Modificación del modelo de datos
- Operaciones sobre los servidores (WAS en nuestro caso)
    - Reinicio de instancia del servidor
    - Reinicio de aplicación
- Solicitud de implantación en el entorno de producción de evolutivos o correctivos

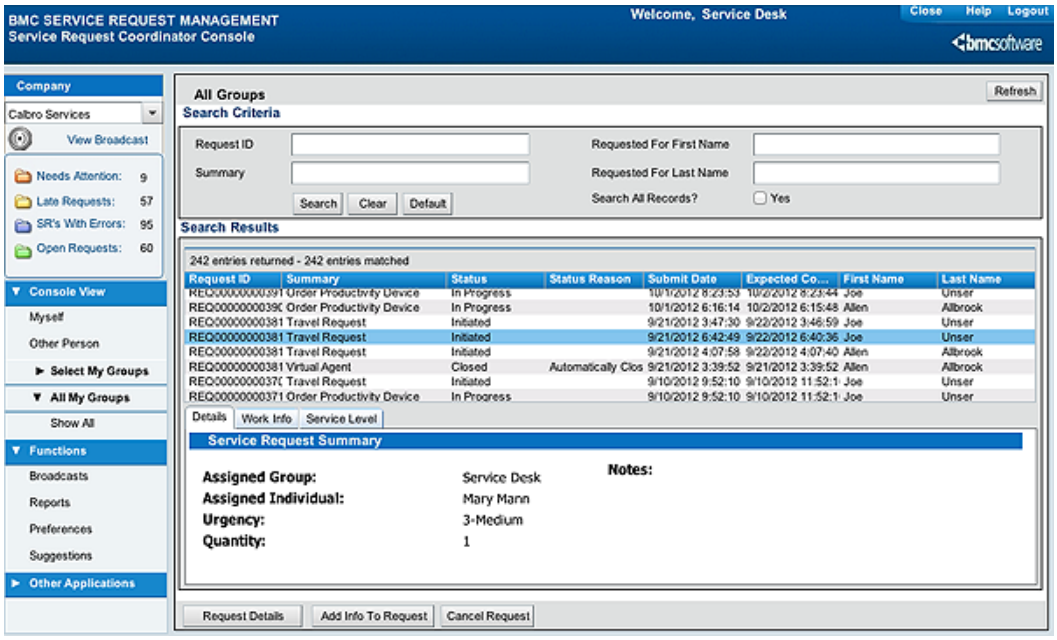

Figura 45. Interfaz gráfica de la aplicación Remedy

- **Herramientas de implantación**

Cuando se termina de integrar en un IDE local el nuevo evolutivo, el siguiente paso es implantarlo por los distintos entornos, hasta que finalmente el software llega a producción.

En este caso disponemos de los siguientes entornos:

Figura 46. Circuito de entornos previos hasta producción

El primer paso es implantar los distintos componentes que conforman el desarrollo en el primer entorno previo, es decir, Desarrollo. En este caso, sólo consiste en conectarnos a la correspondiente máquina de desarrollo, dejar los componentes a instalar en las rutas correspondientes y pedir reinicio de aplicación si se requiere.

Para el resto de implantaciones por el resto de entornos hasta Producción, se usa la herramienta de control de cambios e implantaciones Dimensions, de la empresa Serena.

Esta aplicación, detecta qué componentes han sufrido cambios en los entornos de desarrollo, permitiendo la localización visual rápida de los componentes que se deben implantar.

Además, permite planificar la implantación para un determinado momento, e incluso permitir habilitar o no dichas implantaciones.

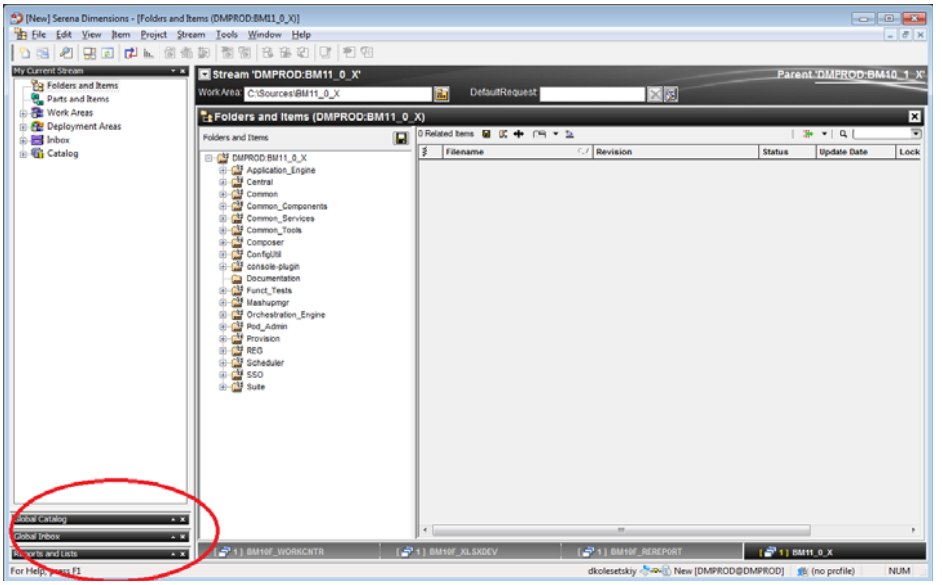

Figura 46. Interfaz gráfica de la aplicación Serena Dimensions

5.3.3. Fases del ciclo de vida del software

En este apartado iremos viendo las etapas que forman es el ciclo de vida de un desarrollo software en esta organización desde la llegada de la metodología LEAN, aunque en el siguiente punto nos centraremos en la etapa de Desarrollo, ya que es en la que más directamente participo.

Posteriormente analizaremos en qué se han visto afectadas tras la entrada de la metodología LEAN.

En esta imagen podemos ver las etapas por las que pasa la gestión de un proyecto software en esta organización desde la llegada de Lean.

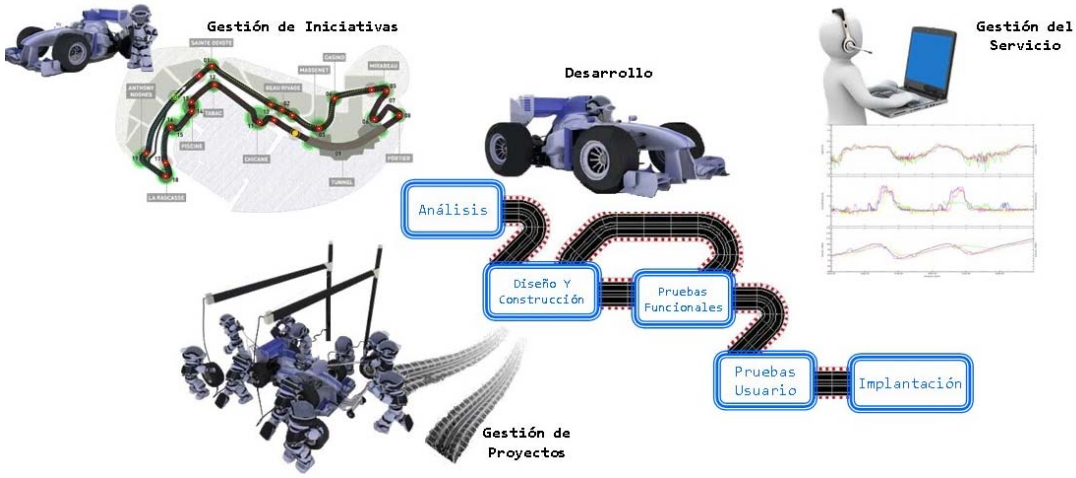

Figura 47. Imagen que representa la metodología seguida

Podemos ver diferenciadas claramente 4 etapas:

Gestión de iniciativas:

Es el proceso por el que se registran las mejoras solicitadas por el usuario, estudiándolas y realizando una valoración temprana del coste del desarrollo. Este proceso lo realiza el responsable del desarrollo del banco, aunque cuenta con el apoyo permanente de DyD sobre todo para realizar la valoración temprana.

Comprende los siguientes puntos:

- Dar de alta la iniciativa con los datos necesarios para poder comenzar a estudiar lo que el usuario está demandando.
- Realizamos un estudio de la iniciativa. Con la información obtenida del estudio, ya estaremos en disposición de realizar una valoración temprana del coste de los desarrollos.
- Aprobar el resultado de la Iniciativa y lanzar el desarrollo.

Gestión de proyectos:

Este es un apartado importante en cuanto a la metodología Lean se refiere, pero transparente para el equipo de DyD.
Aunque todos los proyectos deben pasar por las mismas fases del desarrollo, según la tipología de cada uno será necesario presentar una u otra documentación para que consideren apta la implantación en producción del proyecto.
Para ello, se ha creado un nuevo departamento de Metodología que establece, siguiendo la metodología Lean y en función de las características del proyecto, qué documentación es obligatoria.
Antes de la implantación de Lean, este departamento no existía y todos los proyectos necesitaban la presentación de la misma documentación básica. Estas diferencias se concretarán en el siguiente apartado.

Consta de las siguientes actividades:

- Lanzar proyecto:

  Es el proceso por el que se determinan los aspectos organizativos, funcionales y tecnológicos del proyecto.

  1. Se convoca una reunión con todos los participantes en el desarrollo de la solución, para lanzar el proyecto oficialmente. En esta reunión se comunica cuál es el objetivo del desarrollo, cuál es el alcance, que hitos se han identificado, cómo se va a desarrollar y gestionar, ...

  2. Se debe identificar y resaltar todos los aspectos de organización, coordinación, gestión, dependencias entre las partes,... que se consideren relevantes de resaltar.

- Consensuar el dossier funcional:

  Se consensua el número de documentos mínimos a completar por el proyecto.

  1. Cuando se tenga claro cómo se va a desarrollar el proyecto, su alcance y su objetivo, nos pondremos en contacto con el área de Metodología para consensuar que documentación se va a generar en el proceso de desarrollo, así como áreas de Calidad deben participar en el transcurso del proyecto.

  2. Todo dependerá de las características del proyecto, las cuales se transmitirán al equipo de Metodología para que nos ayuden a configurar el Dossier Funcional y el Ciclo de Vida del Proyecto.

  3. Como resultado de este dialogo se obtendrá un Dossier Funcional validado y aprobado por Metodología que tendrá validez durante todo el proyecto mientras no se realice un cambio significativo de las características del proyecto.

- Planificar y dimensionar el proyecto:

  Asignamos recursos a nuestro proyecto.

  1. En este momento vamos a determinar el número de recursos de los que disponemos y de qué perfil son para poder establecer una planificación inicial del proyecto.

  2. Para trabajar con recursos externos, deberemos seguir el proceso de contratación, emitiendo el pliego de condiciones a las empresas colaboradoras y obteniendo las ofertas de estos que más se ajusten.

  3. Cuando ya dispongamos de los recursos, los asignaremos a nuestro proyecto en la herramienta de gestión, identificando la carga de trabajo que tendrá cada uno a lo largo del proyecto.

Desarrollo:

Es el proceso que realiza directamente el departamento de DyD (diseño y desarrollo) y que comprende las fases desde el análisis de la iniciativa hasta su implantación en el entorno de producción. En esta fase es en la que yo participo más directamente y sobre la que se profundizará para ver cómo la ha afectado la gestión Lean en el siguiente apartado.

Gestión del servicio:

Una vez el proyecto ya está implantado en producción, el usuario nos puede hacer llegar nuevas necesidades.

Consta de las siguientes actividades:

- Recepción y Análisis

  Recoger la necesidad del usuario y registrarla en la herramienta, realizando un análisis para solucionarla.

  1. Cuando llega una necesidad por parte del cliente, la tipificaremos como atención al usuario o como correctivo en el caso de que sea una incidencia.
  2. Se registrará esta solicitud identificando su criticidad y al posible equipo encargado de solucionarla.
  3. Si al analizar la petición vemos que la aplicación afectada no está en nuestro ámbito de trabajo la traspasaremos al equipo correspondiente (ANS, infraestructura…). Hay que tener en cuenta que los proyectos implantados en producción tienen una garantía de un mes de uso por parte del usuario. Una vez pasado ese periodo de garantía, la gestión de incidencias surgidas la debe llevar a cabo el equipo de ANS (Acuerdo Nivel Servicio).
  4. En el caso de tratarse de una incidencia se deberá analizar el impacto en las aplicaciones afectadas y se identificará los componentes que están produciendo el error. Se evalúa el esfuerzo de resolución de la misma y se prioriza en la cartera dependiendo del esfuerzo y su criticidad. En el caso de tratarse de una atención a usuarios, se analizará el coste de resolverla de forma que si se puede resolver fácilmente, se resolverá de forma inmediata.

- Realización

  Atendemos la petición de correctivo o atención a usuarios.

  1. Tanto si se trata de atención a usuarios o correctivos, daremos una respuesta de primer nivel para minimizar el impacto de la incidencia o de la consulta en la medida de lo posible con el objetivo de que el usuario pueda continuar con su trabajo.
  2. En el caso de que la primera respuesta no haya solucionado la petición, analizaremos la causa en profundidad hasta dar con la mejor solución. Finalizado el análisis se implementa la solución a la petición.

3. Si la petición fuese ocasionada por falta de funcionalidad, o la resolución fuese algo más complicada de lo esperado, se podrá abrir una petición de evolutivo para resolverla.

- Cierre

  Implantamos el desarrollo a producción y cerramos la petición.

  1. Cuando hayamos conseguido responder a la atención de usuarios o la incidencia quede resuelta, informaremos al cliente de su resolución y de su inminente cierre en el registro.
  2. En la herramienta de gestión de peticiones, incluiremos todas las acciones que hemos tenido que realizar para su implantación, para que de este modo quede documentado para posibles reincidencias.
  3. Cuando obtengamos el OK del cliente y finalicemos su registro, daremos por finalizada la petición.

5.4. Fase de desarrollo

La fase de desarrollo está compuesta por las siguientes etapas:

- Análisis
- Diseño y construcción
- Pruebas funcionales
- Pruebas usuario
- Implantación

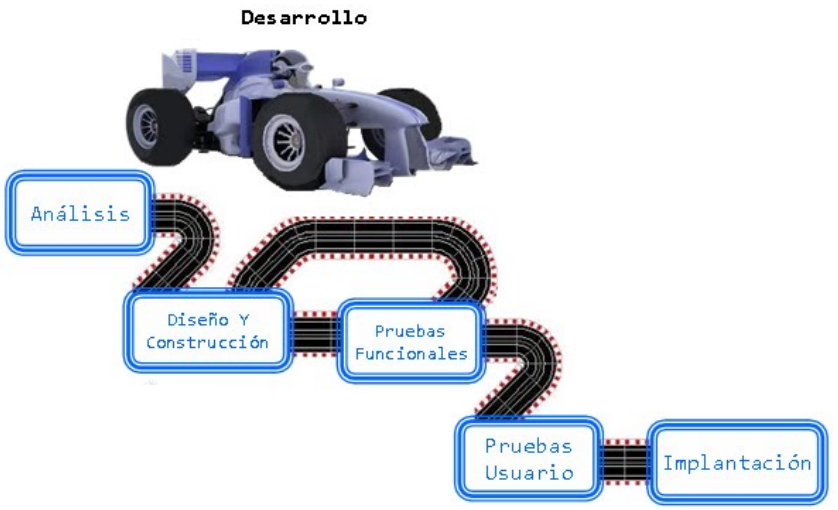

Figura 48. Imagen que representa la fase de desarrollo

5.4.1 Análisis

Esta fase se puede resumir en "¿qué vamos a hacer?".
En ella se pretende detallar la solución a desarrollar para poder valorar con más exactitud el coste del mismo y poder determinar la viabilidad, el alcance así como los procesos afectados.
La fase de análisis se va documentando en un documento denominado "Dossier funcional" y que finalmente será revisando y aprobado por el usuario. Será, por así decirlo, un documento contractual por el cual nos comprometemos con el usuario a desarrollar todas las funcionalidades en él descritas.
Podemos encontrar una plantilla del documento en el Anexo. (A110 DossierdeUsuario).
No obstante, y como bien dice la metodología Lean, este documento y los posteriores, son susceptibles de ser modificados en cualquier momento del desarrollo si el usuario así lo requiere.

También se generan los documentos "Inventario de funcionalidades" donde se van añadiendo las funcionalidades que se ven afectadas (nuevas o modificadas) y un documento de "Descripción de la funcionalidad" donde se detalla cada una de las funcionalidades anteriores.
Podemos encontrar una plantilla de ambos documentos en el Anexo. (A111 - Inventario de Funcionalidades y A112 Descripción de la Funcionalidad).

Participantes:

- Interlocutor con el cliente
- Equipo de desarrollo
- Usuario

Esta fase está formada por las siguientes actividades:

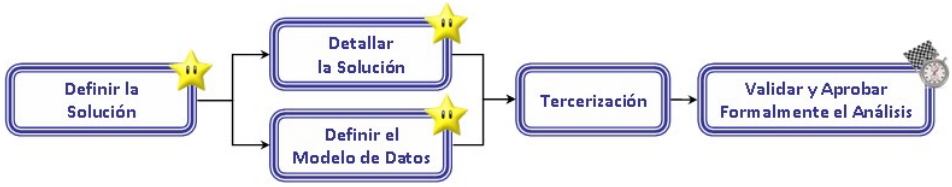

Figura 49. Actividades la fase de análisis

Definir la solución:

Establecer todos los diálogos con el cliente/usuario para detallar lo que nos están requiriendo hacer.

En esta actividad se llevarán a cabo las siguientes acciones:

- Detallar las necesidades funcionales:

Explotar los requerimientos recogidos en la Gestión de la Demanda, identificando más detalle, requisitos no funcionales, impacto en aplicaciones, si implican más requisitos que el usuario no nos ha transmitido…
Las funcionalidades requeridas pueden necesitar modificar o crear otras en la misma aplicación o en otras existentes.
Se va definiendo la estrategia a seguir en las pruebas. En el Anexo encontramos la plantilla que rellenamos (C109 Estrategia y Plan de Pruebas.doc)

- Identificación de perfiles afectados
  Identificar los distintos perfiles y sus características que van a acceder a nuestro desarrollo.

  - Solicitar al Usuario que nos identifique los perfiles que va a necesitar en su aplicación y los niveles de acceso que tendrá cada uno.
  - Si la aplicación es nueva, tendremos que inventariar todos los perfiles que hemos identificado.
  - Si la aplicación ya existe, consultaremos los perfiles actuales que existen en producción y actualizaremos ese registro si hemos detectado algún perfil nuevo.

  En este punto, sobre todo si nos encontráramos con una aplicación nueva, deberíamos ponernos en contacto con el departamento de Seguridad para que nos indicara la manera de proceder.

- Detallar la interfaz de Usuario
  Determinar la apariencia de la aplicación y de todos los interfaces con el usuario.

  - Realizar una maqueta para que el usuario pueda determinar que apariencia va a tener la aplicación, cumpliendo con los estándares de usabilidad definidos.
  - Identificaremos que ventanas van a interactuar con el usuario, mostrando que datos va a ser necesarios que el usuario introduzca en el sistema, detallando tipo de datos, rangos permitidos,…
  - En este momento, se definen las ayudas que se van a proporcionar en la aplicación (que datos se solicitan, para que sirve, como llegar hasta ahí,..)
  - Además se empiezan a elaborar los mensajes de error que se van a mostrar al usuario indicándole la causa y que acción es necesaria realizar para solventarla.

- Definir solución técnica
  Identificar la solución tecnológica más ventajosa para el nuevo desarrollo en el caso de ser necesario.

  - En colaboración con otras áreas se describirá la mejor solución técnica teniendo en cuenta la infraestructura existente, compra de herramientas, la seguridad de la información,..

Esta fase se suele dar en aplicaciones nuevas, o bien, en las nuestras cuando por intentar modernizarlas un poco se han ido introduciendo pequeñas mejoras, como el uso de jQuery, Angular js…

En este caso hay que tener en cuenta a los departamentos de Arquitectura y de ANS, ya que estos últimos serán los encargados de mantener los desarrollos una vez pasado el periodo de garantía.

- Establecer las estrategias del proyecto
  Comenzar a planificar como se va a realizar la conversión, la convivencia con otras aplicaciones, la formación a usuario y la implantación.

  - ¿Cuándo será puesta en producción la nueva solución? ¿Estará completa o se realizará en varias fases? ¿Necesitamos algún componente hardware o software previo que nos condicione?
  - Si la nueva solución va a sustituir a una ya existente, ¿van a convivir o es sustitución inmediata? ¿cómo vamos a eliminar la aplicación existente? ¿durante cuánto tiempo será necesario que estén ambas aplicaciones?
  - ¿Cómo vamos a formar a los usuarios? ¿Cuántos son? ¿Qué estrategia vamos a seguir en la divulgación?

  En esta parte, es cuando se decide algo clave en la metodología Lean y es cuándo el usuario va a ver sus necesidades materializadas. En el caso de nuestros desarrollos se suele tratar de implantar en producción el desarrollo unos 2 meses después de que surgiera la iniciativa. De esta forma el usuario, en su trabajo diario puede valorar si es exactamente lo que necesita y si no lo es pedir las consiguientes adaptaciones.

  Para ello, un evolutivo completo, tal y como se presenta en la iniciativa se suele dividir en releases. Cada release contiene una funcionalidad, o parte de ésta totalmente operativa y que pueda estar lista en aproximadamente 2 meses.
  De esta forma, cada poco tiempo, el usuario tiene listo parte de sus requerimientos y puede validar en real si éstos necesitan algún ajuste. De ser así, se pueden incluir en las releases siguientes.

- Detallar el nivel de servicio de la aplicación
  Identificar las características que deberá cumplir la solución. ¿Cómo debe comportarse?

  - Acordamos con el usuario como se va a comportar la aplicación cuando esté en producción.
  - Determinamos si necesita un tiempo de respuesta especial, si necesita una ventana horaria de respuesta o soporte especial, …
  - Identificaremos si es necesario monitorizar alguna de las funcionalidades planteadas en la solución.

- Definir los criterios de aceptación
  Identificar que tiene que cumplir la aplicación para que el usuario la apruebe.

El interlocutor con el cliente es el encargado de encontrar respuesta a las siguientes preguntas:

- ¿Qué es lo que el usuario necesita comprobar para dar el visto bueno a la aplicación?
- ¿Cuál es el mínimo? ¿Se puede prescindir de algo?
- ¿Qué es lo más importante o crítico para el usuario?

- Aprobación del dossier de usuario
  Una vez finalizado el Dossier de Usuario y aunque ha participado en todo el proceso de elaboración, solicitamos al usuario que nos confirme que todo lo definido es lo que él espera y por lo tanto nos aprobará formalmente el mismo.

Detallar la solución:
Detallar las funcionalidades implicadas en la solución, con las aplicaciones identificadas, tanto las nuevas como la modificación de las ya existentes.

Esta actividad es prácticamente íntegra de nuestro equipo. En ella se detallan las funcionalidades que se ven afectadas, registrándolas en el documento "Inventario de funcionalidades".
Por cada funcionalidad afectada, se genera un documento de análisis "Descripción de la funcionalidad".

En esta actividad se llevarán a cabo las siguientes acciones:

- Identificar las aplicaciones afectadas:
  En esta actividad se trata de dar respuesta a las siguientes preguntas.
  ¿Qué aplicaciones se van a Modificar? ¿Es necesario crear una nueva?

  En nuestro caso, se tratan de modificaciones. En esta actividad hay que tener en cuenta cómo cambios en nuestra aplicación impactarían en otras. Normalmente suele ser porque se comparten diversos datos entre ellas.

- Identificar las funcionalidades de la solución
  ¿Qué funcionalidades se van a crear, modificar o eliminar de cada una de las aplicaciones afectadas?

  - Entendemos por funcionalidad cada una de las operativas que se puede realizar desde una aplicación. Bien por el usuario de forma on-line como todo lo que la aplicación tenga que realizar en procesos back.
  - Para cada una de las aplicaciones afectadas, identificar las funcionalidades a crear o modificar identificándolo.

- Detallar la funcionalidad
  Analizar cada una de las funcionalidades afectadas, llegando al mayor nivel de detalle posible.

  - Para cada una de las funcionalidades identificadas, detallarla indicando cuál es su objetivo, todos los pasos a realizar en ella, que entidades requiere para

realizar cada uno de los pasos, si requiere de un tercero, que entradas necesita y que salidas genera...
  - Cuando consideremos que la funcionalidad está ya con el suficiente detalle, indicaremos que está lista para ser revisada.
  - Asociaremos a cada funcionalidad los perfiles que van a poder acceder a la misma.

- Validación del inventario de funcionalidades
  Se valida principalmente, que cada funcionalidad afectada y añadida al "Inventario de funcionalidades" tiene su documento asociado de "Descripción de la funcionalidad"

Definir el modelo de datos:

Describir los datos que van a necesitarse en la solución y como se van a relacionar entre ellos.

En esta actividad se llevarán a cabo las siguientes acciones:

- Definir modelo lógico de datos:
  Usando la normalización, definir las tablas y sus relaciones necesarias para la solución.

  - En el Modelo Lógico de Datos se detallada cada una de las diferentes entidades, relaciones existentes entre ellas así como su cardinalidad, claves primarias y atributos principales.
  - Una vez diseñado el Modelo de Datos, se normaliza hasta la tercera forma normal, se comprueba la ausencia de reiteraciones e incongruencias y se verifica el modelo.

  El modelo lógico se define con la herramienta PowerDesigner.
  Nuestras aplicaciones al ser antiguas, no usaban modelo lógico si no directamente el físico por lo que este paso nos lo saltamos y el siguiente también.

- Validación modelo de datos:
  Revisar si el modelo lógico cumple la normativa existente y está correctamente normalizado.
  Cuando se terminaran de modelar los datos, OCTA comprobaría si cumple con todos los estándares definidos en el modelado de datos.

Gestionar la tercerización de la solución:

Actualmente la construcción se realiza de forma externa en factorías de software, aunque pequeñas modificaciones aún las seguimos realizando directamente con nuestro equipo.
De hecho existen dos tipos de factorías:

- Factoría de construcción: son los que codifican nuestro diseño documentado
- Factoría de pruebas: llevarían a cabo la fase de pruebas, pero nuestras aplicaciones no usan este tipo de factorías.

En esta actividad se trata de comunicar al departamento de Factorías que necesitamos unos nuevos desarrollos, para así ellos gestionar qué factoría será asignada según su carga de trabajo.

Digamos que se realiza una reserva a factoría. En ella mediante una aplicación interna del banco se detalla el número de componentes implicados en el desarrollo, tipo y si son nuevos o modificaciones. En el Anexo, podemos ver un ejemplo de plantilla de documento para hacer la reserva a factoría (F213 Inventario de Entregas a SwF Java)

Validar y aprobar formalmente el análisis:

Obtener la aprobación del análisis, tanto de las áreas técnicas implicadas como del cliente.

En esta actividad se llevarán a cabo las siguientes acciones:

- Comprobar la calidad:
  Comprobar que se han tenido en cuenta todos los requerimientos iniciales, que todo está correctamente documentado y que no se ha omitido ningún aspecto técnico o funcional…

  - Antes de dar por cerrada la fase de análisis, revisaremos el resultado obtenido a lo largo de este tiempo. ¿Está listo para dar el siguiente paso? ¿Sería conveniente que Metodología comprobará si esta todo? ¿Se han realizado todas las revisiones formales?

- Informar al equipo de mantenimiento (ANS):
  Transmitir al equipo de mantenimiento todo el conocimiento adquirido junto a la documentación generada durante la fase de análisis.

  - Informaremos al equipo ANS´s de lo que vamos a hacer y para cuándo estará listo.
  - ¿Prevén tener que tocar algo de las aplicaciones que van a ser modificadas?
  - ¿Están estables como para admitir más cambios?

- Concretar la valoración del proyecto:
  Completar la valoración en horas y coste del proyecto de forma más exacta.

  - Con todos los datos de los que disponemos, ahora podemos hacer una valoración más detallada de lo que cuesta el proyecto.
  - Si hay desviaciones, negociar si es necesario modificar el alcance del proyecto, o se alarga en plazo/coste.

  Se detallaría en el “Documento de valoración”

- Aprobación formal del cliente
  Obtener la certificación y aprobación del cliente de esta Fase

- Con todo el nivel de detalle que disponemos, solicitaremos al cliente que nos apruebe la continuidad del proyecto.
- Es importante si hay dudas o cuestiones que no estén aceptadas, no continuar con el proyecto para evitar de este modo tener que rehacer trabajo o realizar trabajo que luego se desperdicie

5.4.2 Diseño y construcción

Esta fase se puede resumir en "¿Cómo lo vamos a hacer?... y lo hacemos".
En ella se desarrolla el detalle técnico de los componentes que formarán parte de la solución y su posterior construcción.
El equipo de proyectos es el encargado de tener lista la infraestructura necesaria en la máquina de desarrollo para poder instalar posteriormente el desarrollo construido, así como de tener listas las oportunas modificaciones de base de datos.
Además, se prepara toda la documentación del diseño y se transmite a la factoría pertinente. Durante la construcción por parte de la factoría se establece continua comunicación con ellos resolviendo las dudas y modificaciones que puedan surgir.

Una vez factoría termina la construcción, nos encargamos de integrar sus componentes e instalarlos en la máquina de desarrollo para pasar una primera tanda de pruebas, principalmente dirigidas a la integración.

Se generan los documentos "Inventario de componentes" donde se van añadiendo las componentes que se crearan / modificaran asociándoles la funcionalidad a la que pertenecen y un documento de "Descripción del componente" donde se detalla la información necesaria para que el componente sea construido.
En el Anexo, podemos encontrar las plantillas de los documentos (D110 - Inventario de Componentes y D111-DetalledelComponente)

Participantes:

- Interlocutor con el cliente
- Equipo de desarrollo

Esta fase está formada por las siguientes actividades:

Figura 50. Actividades la fase de diseño y construcción

Diseño físico de estructuras de datos:

Realizar el diseño físico de datos y su estructura partiendo de la definición lógica del Análisis.

En esta actividad se materializan las modificaciones de base de datos que se detectaron en la fase de análisis.

Para ello mediante petición por la aplicación Remedy se solicitan las modificaciones necesarias adjuntando el modelo de datos previamente modificado con la herramienta PowerDesigner.
Al solicitar tablas nuevas, se debe tener en cuenta la volumetría de datos que se espera y el crecimiento anual estimado de la tabla.

Preparación de entorno de desarrollo:

Realizar todas las tareas necesarias para preparar los entornos previos.

Se solicita mediante peticiones Remedy al área correspondiente las acciones necesarias para la preparación de los entornos y verificar que estos se encuentran tal y como se requieren.

- Debemos asegurarnos de tener todo el soporte tecnológico listo para poder disponer de un entorno estable, por ejemplo, si se requieren nuevos directorios…
- Instalamos en nuestro entorno el software necesario. Puede ser necesario también modificar el classpath de la aplicación para incluir nuevas librerías…
- Preparamos los juegos de datos que vamos a necesitar para las pruebas.

Diseño detallado:

Preparar la documentación necesaria para aportar a los programadores y poder realizar la codificación de la implementación necesaria.

En esta actividad se llevarán a cabo las siguientes acciones:

- Completar el inventario de componentes:
  Identificar los componentes físicos que van a montar la solución.
  - Partiendo de las funcionalidades identificadas en la fase de análisis, identificar que componentes van a ser necesarios crear o modificar para implementar la solución.
  - Cada componente quedará relacionado con una funcionalidad.
  - En el caso de que se vaya a construir en Factoría de Construcción, se comienza a planificar las entregas que se van a realizar a la factoría, indicando a que paquete de entrega va cada componente.

- Diseño detallado del componente
  Diseñar y detallar cada uno de los componentes identificados en el inventario incluyendo los requisitos y los pasos que se deben seguir en su ejecución.
  - Completar toda la información necesaria por componente para que pueda ser construida.
  - El nivel de detalle a alcanzar en cada componente, dependerá de la necesidad de tener que enviarlo a Factoría de Construcción.

- Toda la documentación se mantendrá referenciada en el Inventario de Componentes lo que permitirá garantizar la trazabilidad.

Nuestras aplicaciones son aplicaciones J2EE por lo que como norma general, los componentes a diseñar son clases java o jsp's.
Actualmente los diseños los estamos generando de la siguiente manera.

Si el componente se trata de un jsp:
- De detalla su diseño en un documento "Detalle del componente"
- El diseño se apoya en el prototipo que se hizo en la fase de análisis.

Si el componente se trata de una clase java:
- El diseño de todas las clases java se detalla mediante UML
- Se genera un proyecto UML por cada funcionalidad afectada. En cada UML se generan, al menos, los siguientes diagramas:
  - Diagrama de casos de uso explicando la funcionalidad
  - Diagrama de clases lógico. En cada clase lógica es donde se detallan las acciones a seguir para codificar el componente.
  - Diagrama de clases físico generado automáticamente por la herramienta, generando a su vez las clases físicas.

- Tipificar y revisar componentes
  Tipificar los componentes para su envío a factoría de construcción identificando el nivel de dificultad y complejidad.

  - Se revisa la descripción realizada del componente con el objetivo de asegurar que se cumplen con los estándares definidos y que están suficientemente detallados, sobre todo si se va a construir el componente en Factorías.
  - Una vez comprobados y revisados, si se va a construir en Factorías, se tipifican cada uno de los componentes en la hoja tipificación de la entrega en la que se ha incluido el componente. Se utilizará la hoja de tipificación de la plataforma en la que se va a construir. En nuestro caso Java. En el Anexo podemos encontrar una plantilla de la hoja de tipificación (F013 - Hoja de Tipificacion Java)

<u>Diseñar casos de prueba:</u>

Diseñar los casos de prueba a los que será sometido el sistema antes de su implantación.

En esta actividad se generaran diversos documentos de "Casos de prueba". En general, se genera un documento de casos de prueba por cada funcionalidad afectada. En el Anexo, encontramos la plantilla de este documento (C204 Casos de Prueba)

En esta actividad se llevarán a cabo las siguientes acciones:

- Diseñar casos de prueba funcionales y de regresión
  Diseñar los casos de pruebas funcionales y de Regresión necesarios para verificar el funcionamiento del sistema y que este cumpla los requisitos que se habían establecido.

  - El Equipo de Desarrollo se encargará de definirlos y documentarlos con la suficiente cobertura y profundidad para asegurar la calidad esperada en la ejecución de las pruebas.
  - La documentación generada en el diseño de los casos de prueba, se debe mantener trazada con las funcionalidades identificadas en el análisis, para poder reutilizar los casos de prueba en futuras evoluciones de los desarrollos.

- Diseñar casos de prueba de aceptación
  Diseñar los casos de prueba que permitirán al usuario aceptar la solución desarrollada.

  - En este momento, se definirán todos los casos de prueba necesarios para que el usuario pueda comprobar que se ha construido lo que esperaba así como el correcto funcionamiento. Estas pruebas deben tener visión usuario, se parte de lo que el usuario ha pedido. No es necesario entrar a detalle de como se realizan las acciones, sino de que se muestra el dato correcto y de cómo se muestra.
  - Se pueden reutilizar casos de prueba definidas como funcionales, aunque no deberían de tener la misma cobertura. Las pruebas de aceptación de usuario deberían ser menos.

Construir el código:

Obtener el código de cada uno de los componentes de la solución.
En esta fase, se realiza en envío a factoría de los componentes diseñados por el equipo de proyectos. Mientras factoría construye, se les da soporte atendiendo a las dudas que puedan sobre el diseño o realizando modificaciones en él que puedan surgir.

En esta actividad se llevarán a cabo las siguientes acciones:

- Envío de diseños
  Se envía a factoría de construcción lo siguiente:

  - Hoja de tipificación de los componentes que deben construir. En ella se detalla qué componentes son nuevos y cuáles modificados y la complejidad de cada uno. Es en función de esa complejidad por lo que se calculará el coste de la factoría.
  - Documentación de diseño
    - Detalle de componentes
    - UML's
  - Componentes a crear / modificar. En el caso de nuevos métodos la signatura del método ya va creada.
  - Resto de componentes necesarios para que las funcionalidades a desarrollar funcionen.

  Toda esta documentación queda alojada en el recurso compartido para que sea accesible a todas las partes implicadas.

- Construir el código
  Durante esta fase, factoría codifica los componentes definidos en nuestros diseños. Tanto ellos en su construcción, como nosotros en el diseño hemos debido tener en cuenta los estándares y las normas definidas.

  En esta fase, se atienden las posibles dudas surgidas por la factoría y posibles modificaciones que se vean necesarias y que no hayan sido consideradas en el diseño.

- Verificar la calidad del código
  Verificar que el código generado en entorno de desarrollo cumple los estándares de codificación de la instalación orientado al rendimiento del programa en producción.

  Cuando factoría nos devuelve el código construido, lo integramos usando la herramienta RAD y generando las correspondientes versiones de código mediante ClearCase.

  Pasaremos el analizador de código PMD para validar que el código construido cumple las reglas mínimas establecidas por el banco.

Ejecutar pruebas unitarias y de ensamblaje:

Comprobar que el código no errores de codificación y en el caso de detectar errores de codificación, gestionar su corrección.

En la construcción con factoría, ellos son los encargados de ejecutar las correspondientes pruebas unitarias de manera que a nosotros nos entreguen el código libre de errores de compilación y asegurándonos que los componentes funcionan individualmente y que interaccionan entre ellos correctamente.

Cierre diseño y construcción:

Nos aseguramos que el 100% del código está construido y listo para poder probarlo funcionalmente.

5.4.3 Pruebas funcionales

Esta fase se puede resumir en “Probamos que funciona”.
En ella se comprueba que la solución desarrollada no tiene errores de diseño y que contempla todas las funcionalidades incluidas en la solución.

Esta fase está formada por las siguientes actividades:

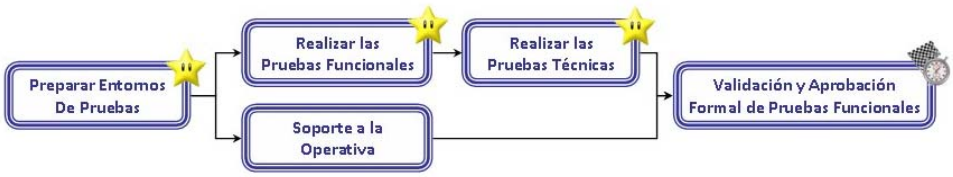

Figura 51. Actividades la fase de pruebas funcionales

Preparar Entornos de Pruebas:

Realizar todas las tareas necesarias para preparar todos los entornos previos donde se va a realizar pruebas a la solución.

Las tareas pueden consistir en crear rutas, ficheros de pruebas, modificar ficheros de configuración, preparan inicializaciones y cargas de datos iniciales…

Como hemos comentado en diferentes apartados, trabajamos con los siguientes entornos previos:

- Desarrollo
- Integración
- Aceptación de Usuarios

Inicialmente nuestras pruebas funcionales comenzarán en el entorno de desarrollo, ya que en el caso de tener que corregir algún componente, en este entorno tenemos disponibilidad total para instalar sin usar la aplicación Dimensions.

Realizar las pruebas funcionales:

Ejecutar todas las Pruebas Funcionales definidas para poder asegurar que la solución cumple con lo esperado tanto a nivel funcional como no funcional (estabilidad, rendimiento,...).

Ejecutaremos en el entorno de desarrollo los casos de pruebas definidos en anteriores fases. Tenemos que verificar:

- que la solución cumple con los requerimientos funcionales identificados.
- el correcto funcionamiento de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.
- que las modificaciones llevadas a cabo sobre la aplicación no afectan al resto de funcionalidades de la aplicación, aunque no se hayan modificado.

Una vez comprobado que el entorno donde se van a ejecutar las pruebas está listo y con todo lo necesario para poder ejecutar las pruebas, se ejecutan las pruebas previamente definidas.

Es importante registrar las incidencias detectadas y su corrección posterior para poder agilizar la corrección de posibles incidencias similares posteriormente en otros entornos.

Una vez realizadas las pruebas funcionales diseñadas para la solución, es necesario realizar las pruebas de regresión para poder afirmar que la implantación de la nueva solución no tiene impacto en otras funcionalidades o aplicaciones.

Valoración y aprobación formal de las pruebas funcionales

Cierre de la fase de Pruebas Funcionales, asegurando desde este momento que la solución está lista para ser probada por el usuario

- Cuando se hayan corregido todas las incidencias y el desarrollo de la solución esté listo para ser probado por el usuario, validaremos y daremos por finalizada la fase de pruebas funcionales.
- Se implantarán todos los componentes en el entorno de integrado y posteriormente en el de aceptación de usuario, ya que es ahí donde el usuario ejecuta sus pruebas mediante la herramienta Dimensions.
- En el caso de que se haya tenido que modificar algún componente, nos aseguraremos de que en el entorno de aceptación de usuario están los componentes en su última versión

5.4.4 Pruebas de Usuario

Esta fase se puede resumir en "¿Es lo que se quería?".
En ella el usuario realiza sus pruebas para validar que satisface todos sus requisitos establecidos.

En esta fase, el equipo de proyectos es el encargado de preparar el entorno de aceptación de usuarios para las pruebas de éste. En este entorno, periódicamente se realizan cargas de datos directamente de producción (con los datos que afectan a la Ley Orgánica de Protección de Datos encriptado), por lo que el usuario encuentra este entorno muy similar a Producción.
Además, estará atento a solucionar todas las incidencias que pueda ir encontrando el usuario en sus pruebas.

Nuestro responsable junto con el usuario, son los encargados además de recoger en el Manual de Usuario toda la operativa que se puede realizar sobre estas nuevas funcionalidades.

Esta fase está formada por las siguientes actividades:

Figura 52. Actividades la fase de pruebas de usuario

Solicitar implantación en Aceptación de Usuarios:

Solicitar la implantación de los componentes y la carga de datos en el entorno de Usuario.

- Identificamos todos los componentes que queremos implantar en el entorno, y para ello debemos asegurarnos que seleccionamos la versión adecuada, que no vamos a generar inconsistencias en el código.
- Solicitaremos la implantación en el entorno de Usuario de los componentes y de todas las estructuras de datos y ficheros de configuración y maestros necesarios, así como la ejecución de los scripts necesarios para la adecuación del entorno.
- Para adecuar el entorno será necesario solicitar la carga de datos iniciales para poder hacer las pruebas necesarias.
- Si se considera necesario, podremos pedir a Seguridad que nos replique de Producción los distintos perfiles para las Pruebas de Usuario.

Realizar pruebas de Usuario:

El usuario comprueba que el resultado de las pruebas es el esperado.

- Una vez comprobado que el entorno donde se van a ejecutar las pruebas está listo y con todo lo necesario para poder ejecutar las pruebas, se ejecutan las pruebas previamente definidas.
- El objetivo de estas pruebas es que el usuario compruebe que lo que solicito y que está recogido en el Dossier de Usuario es exactamente lo que se ha desarrollado y que el funcionamiento es el esperado.
- Es importante registrar las incidencias detectadas y su corrección posterior para poder agilizar la corrección lo antes posible.

Validar y aprobar formalmente las pruebas de Usuario:

Aprobar las Pruebas realizadas durante esta fase, verificando los niveles definidos en la fase de Análisis

5.4.5 Implantación

Esta fase se puede resumir en “Lo ponemos en producción”
En ella se realiza la implantación de los programas, módulos, componentes y demás elementos que forman el sistema, y que han sido diseñados, construidos, probados y validados en las fases anteriores.

Esta fase está formada por las siguientes actividades:

Figura 53. Actividades la fase de implantación

Ejecutar Pruebas de Pre-implantación:

Ejecutar en el entorno previo a producción, pruebas anteriores a la subida para verificar que la misma se hará con eficacia.
Debemos poner nuestra atención en los puntos críticos que hemos identificado a lo largo del desarrollo.
Debemos comprobar que todas las incidencias detectadas se han solucionado. Adicionalmente, realizaremos pruebas de regresión en las aplicaciones impactadas para asegurar que su funcionamiento no se ha visto afectado.

Puesta en producción:

Realizar la implantación del sistema y de todos sus componentes en el entorno de producción.

En esta actividad se llevarán a cabo las siguientes acciones:

- Preparar el entorno de producción
  Gestionar la preparación del entorno de producción de acuerdo con lo definido en los planes de implantación del desarrollo.

    - Debemos solicitar el perfilado necesario para que esté listo el día de la implantación.
    - Debemos solicitar el soporte necesario para el día de la implantación.
    - Confirmar que tenemos los medios y puestos de trabajo disponibles para el día de la implantación.

- Solicitar el paso a producción
  Finalizar la instalación de los componentes físicos y lógicos del entorno y asegurar que los elementos técnicos y los recursos humanos necesarios para el paso a producción del sistema estén disponibles.

    - Realizamos las solicitudes Remedy necesarias para poder implantar el sistema al departamento de Recepción de Aplicaciones (RA).
    - Debemos chequear que no olvidamos ningún componente.
    - Y finalmente liberamos del entorno los desarrollos que se van a implantar.

- Subida a producción
  Finalizar la subida a producción de los componentes por parte de los equipos técnicos.
    - Revisamos que la subida es correcta y que se han implantado todos los componentes.
    - Y comprobamos que la ejecución de los programas es correcta.

5.5. Conclusiones de la aplicación de la metodología LEAN en el ciclo de vida del software

Volviendo a los 7 principios básicos de la metodología LEAN paso a describir en qué manera nos han afectado en el desarrollo de software.

1. Eliminar desperdicio

Brindar un liderazgo técnico y de mercado.
Aunque las aplicaciones que gestionamos pueden considerarse como un poco anticuadas, en los nuevos evolutivos se nos ha animado a ir integrando nuevas soluciones tecnológicas, sobre todo en cuanto a mejorar la interfaz se refiere.
Esto al usuario le ha entusiasmado y continuamente nos está demandando adaptar más partes de la aplicación mejorando la interfaz.
De esta manera, el usuario ve que el equipo de proyectos es capaz de producir productos innovadores y tecnológicamente avanzados.

Crear sólamente cosas de valor
El recién creado departamento de Metodología es el encargado de decidir qué documentación es necesaria o no.
No tiene sentido que en pequeñas modificaciones se tengan que seguir los mismos documentos que en el desarrollo de un módulo nuevo en la aplicación.
Además, tenemos que detectar qué requerimientos del usuario podemos decir que sobran ya que no aportan valor a la aplicación y hacérselo ver.
También tenemos el caso contrario. A veces, para que “quede bonito” se nos ocurren pequeños detalles o modificaciones. Tenemos que valorar si son necesarias y aportan realmente valor, porque suele ser en esas pequeñas modificaciones cuando surgen más problemas e incidencias al considerarlas triviales y hacer menos hincapié en sus pruebas.

Escribir menos código
Mientras más código se tenga, más pruebas se van a necesitar, por lo que se necesitará más trabajo.

2. Crear conocimiento

Crear equipos de diseño y construcción

Se pretende trabajar siempre con los mismos equipos de proyectos. De esta forma, los equipos tienen una gran experiencia sobre la organización y la aplicación en la que trabajan, haciendo más fáciles en particular las fases de análisis y diseño.
Un equipo con gran experiencia sobre la aplicación es un equipo mucho más resolutivo y adaptable a los cambios.

Además, se ha ido un paso más allá. Se ha creado un sistema para intentar trabajar siempre con los mismos proveedores (consultoras). Mediante un sistema de puntos se premia sobre todo a las que más conocimiento tienen sobre la organización.

El líder del equipo de desarrollo, responsable del banco, tiene que escuchar a los miembros y hacerles preguntas inteligentes que los inste a buscar respuestas y volver lo más pronto posible con los problemas que surgen, o con las soluciones inventadas.

Mantener una cultura de mejora continua

Hay un ambiente donde las personas están mejorando continuamente en lo que trabajan. Sabemos que no somos (y no debemos ser perfectos) y que siempre tienen algún área que pueden mejorar.

Enseñar métodos de resolución de problemas

Los equipos de desarrollo deberían comportarse como pequeños centros de investigación, estableciendo hipótesis y realizando varios experimentos rápidos para verificar su validez.

3. **Calidad integrada**

Mayor hincapié en las pruebas
Cuanto más esté probado un producto software menos susceptible es a provocar incidencias en el entorno de producción.

Automatizar tareas rutinarias
Se intentan automatizar las pruebas, la construcción, las instalaciones, y cualquier cosa que sea rutinaria. Hay que automatizar de una manera inteligente, de forma que las personas puedan mejorar el proceso y cambiar cualquier cosa que quieran sin preocuparse por si el cambio hace que las cosas dejen de funcionar.

4. **Decidir tan tarde como sea posible**

Mantener las decisiones irreversibles hasta el último momento
Hasta no estar bien avanzado el desarrollo no quedar con al usuario en una fecha en concreto de implantación.
Si tenemos dudas en cómo afrontar un problema, mantener varias soluciones para todas las decisiones críticas y ver cuáles funcionan mejor.

5. **Optimizar el total**

Enfocarse en el flujo completo de valor
No hay que gastar esfuerzo en optimizar ineficiencias locales, sino en ver el todo y optimizar a la organización en su totalidad.

6. **Entregar rápido**

Dividir los requerimientos del usuario en releases
Dividir funcionalidades que puedan funcionar de forma independiente e ir desarrollando e implantándolas de manera que cada poco tiempo

(aproximadamente 2 meses) el usuario pueda verlas en funcionando en el entorno real de producción y así detectar posibles incidencias o mejoras.

Esto facilita además la planificación del proyecto, la gestión de la asignación de recursos es más fácil planificarla para plazos relativamente cortos.

Los desarrollos se hacen más dinámicos. La tarea varía constantemente.
En los desarrollos largos puedes estar elaborando casos de pruebas durante un mes, con lo tediosa que resulta esa tarea…

Anteriormente a la metodología LEAN, existía un departamento de negocio que servía de intermediario entre el departamento de diseño y desarrollo correspondiente, y el cliente. Este departamento gestionaba con el usuario las necesidades que iba encontrando agrupándolas en proyectos.
Normalmente, durante un año al equipo de diseño y desarrollo nos llegaban uno o dos proyectos, con su correspondiente documento de requisitos. Esto implicaba, que se intentaban realizar a lo sumo dos implantaciones al año.
Este planteamiento daba lugar a situaciones en las que el usuario definía sus requerimientos a principios de año y casi a finales de este encontraba su necesidad materializada. Ocurría que a veces, cuando la funcionalidad estaba implantada ya no era útil. Al usuario, le habían surgido otras necesidades que sustituirían a la anterior, le resultaban incompletas, etc. Esto se traducía en trabajo desaprovechado.

7. Respetar a las personas

Capacitar a los líderes de equipo
Darles a los líderes de equipo entrenamiento, guías y espacio libre para implementar el pensamiento Lean en su ambiente.

Mover la responsabilidad y la toma de decisiones al nivel más bajo posible
Dejar que el equipo de proyectos tome más decisiones (ellos saben mejor que nadie cómo implementar algoritmos difíciles y aplicar tecnologías de última generación), además conocen la aplicación.

Fomentar orgullo por el trabajo
Fomentar la pasión y la participación del equipo hacia lo que hacen y cómo lo hacen. Esto elimina el individualismo, ya que se promueven los logros colectivos frente a los individuales.

Fomentar la flexibilidad laboral
Los trabajadores vemos la flexibilidad laboral como un incentivo. Tener trabajadores contentos implica menos rotación de personal, por lo que tendremos gente con experiencia trabajando en el equipo y con noción de formar pertenencia a un equipo.

Pero de hecho, hay que tener en cuenta que todo sigue siendo mejorable, y es que claro está, hay que tener en cuenta la mejora continua de la que tanto se habla en la metodología LEAN.

En cuanto al uso de herramientas, en este trabajo sólo se han citado las que actualmente estamos usando, pero nos consta que la organización está valorando incluir el uso de herramientas que apoyen más la metodología LEAN.

Bien es cierto que las aplicaciones que gestionamos son algo antiguas y el coste de adaptarlas para poder trabajar con ellas podría ser bastante elevado, pero quizás poco a poco se intenten aplicar algunas.

El uso de estas herramientas estaría sobre todo encaminado a automatizar de manera eficiente la integración continua. La situación ideal sería la mostrada en la figura 30.

Actualmente, “simulamos” la integración continua, pero es una tarea dependiente de las personas. No tenemos ningún proceso automatizado, por lo que depende de la experiencia y de nuestro buen hacer el tener todos los días el código actualizado desde el repositorio, el subir esas modificaciones en todos los entornos previos.

Además sería muy interesante el poder automatizar de alguna manera diversas pruebas. Actualmente es una tarea que consideramos muy tediosa y en la que debemos ser más eficientes, ya que las pruebas las construimos y ejecutamos las mismas personas que diseñamos el sistema. Esto creo que nos hace ser menos rigurosos en ellas.

En general podemos decir que la organización está muy comprometida en general con la metodología LEAN en su concepción general y que poco a poco la está implantando en sus distintos departamentos.

Ha creado varios grupos de trabajo sobre metodología LEAN en los que compartir experiencias y diversos blogs y wikis, y como he dicho anteriormente nos consta que se está trabajando para incorporar nuevas herramientas que se recomienda en la gestión LEAN.

6. Conclusiones generales

La literatura revisada pone de manifiesto el creciente interés en los últimos años por extrapolar las técnicas y prácticas de la filosofía Lean a la Ingeniería del Software.
La aplicabilidad de las técnicas y prácticas Lean en esta disciplina se pone de manifiesto y los resultados obtenidos en las experiencias son notablemente positivos.
El enfoque predominante de los estudios es la extrapolación de un concepto o práctica al ámbito de la Ingeniería del Software siendo escasas las referencias que abordan el problema de una manera integral.
El número de estudios prácticos sobre el efecto en proyectos reales es muy reducido, aunque el éxito de los resultados obtenidos han empujado a las empresas que pilotaron experiencias Lean a extender la práctica al resto de la organización.
Las sinergias que se observan con algunas de las prácticas actuales en Ingeniería del Software, en particular con las prácticas ágiles, hace pensar que en la medida que aparezcan nuevas evidencias del éxito en la adopción de esta filosofía servirán de catalizador de nuevas experiencias.
Existen factores en el escenario actual para apostar por la adopción de los principios y prácticas de esta filosofía:

- Dificultades para la financiación
- Contracción de la demanda (menor que la oferta)
- Emergencia de nuevos modelos de negocio
- Mayores exigencias de los consumidores
- Necesidad imperiosa de mejorar la productividad

A continuación se exponen las debilidades, amenazas, fortalezas y oportunidades que se concluyen del estudio realizado.

- **Debilidades**
    - El número de estudios que abordan la implementación de la metodología Lean en el ámbito de la Ingeniería del Software en empresas reales es muy reducido, no abordan una implementación integral y en ningún caso pueden considerarse como una guía práctica de implementación, a lo sumo, un conjunto de buenas prácticas.
    - Faltan estudios y ejemplos reales que avalen los resultados de Lean en este ámbito complica la obtención de la aprobación de la Dirección de una organización para la adopción de esta filosofía.
    - El componente cultural puede ser un obstáculo para el éxito de la implantación de esta filosofía, en especial la necesidad de disponer de un estado de mutua confianza empresa – empleado.

- **Amenazas**
    - Falta de aceptación por parte de los individuos de la organización. El componente motivacional
    - Esperar resultados a corto plazo. Es frecuente en las empresas del sector esperar resultados que demuestren la efectividad del cambio a

corto plazo, algo que estará en contra de la necesaria visión a largo plazo de Lean.
- La emergencia de fenómenos como la deslocalización (offshoring, nearshoring) y aproximaciones Tayloristas basadas en fuerza bruta y menores costes salariales van en contracorriente respecto al criterio de motivación y excelencia propuesto por la filosofía Lean.

- **Fortalezas**

  - Todas las experiencias que se han encontrado de la aplicación de Lean a la Ingeniería del Software ofrecen resultados positivos y pueden servir de argumento para realizar nuevas experiencias.
  - La alineación de la filosofía Lean con los principios ágiles, mucho más arraigados en el sector, pueden servir de catalizador de nuevas aproximaciones Lean.

7. Bibliografía

- Sommerville, Ingeniería del software, 7º ed. Madrid: Pearson Educación, 2005
- R. Pressman, Ingeniería del software: un enfoque práctico. Madrid: McGraw-Hill, 2001
- C. Larman, Agile and iterative development: a manager's guide. Boston: Addison-Wesley, 2004.
- J. Liker, The Toyota way: 14 management principles from the world's greatest manufacturer. New York: McGraw-Hill, 2004.
- J. P. Womack y D. T. Jones, Lean Thinking: Banish Waste and Create Wealth in Your Corporation, Revised and Updated, 2º ed. Free Press, 2003.
- P. Deemer y G. Benefield, "The Scrum Primer. An Introduction to Agile Project Management with Scrum," 2007.
- L. Wilson, How To Implement Lean Manufacturing, 1º ed. McGraw-Hill Professional, 2009.
- E. Parnell-Klabo, "Introducing Lean Principles with Agile Practices at a Fortune 500 Company," in AGILE 2006
- M. Poppendieck y T. Poppendieck, Lean Software Development: An Agile Toolkit. Addison-Wesley Longman Publishing Co., Inc., 2003.
- M. Poppendieck y T. Poppendieck, Implementing Lean Software Development: From Concept to Cash (The Addison-Wesley Signature Series). Addison-Wesley Professional, 2006.
- C. Hibbs, The art of lean software development. Sebastopol CA: O'Reilly Media, 2009
- H. Kniberg, Kanban and Scrum - making the most of both. Lulu.com, 2010.
- Agile Project Manifesto: www.pmdoi.org
- Web de Poppendieck: http://www.poppendieck.com/
- Blog de Poppendieck: http://www.leanessays.com/
- Blog sobre desarrollo de software LEAN: https://leansoftwaredevelopment.wordpress.com/
- Grupos de desarrollo ágil:

http://www.agileleadershipnetwork.org/
https://groups.yahoo.com/neo/groups/agileprojectmanagement/info

- Manifiesto por el Desarrollo Ágil de Software: http://www.agilemanifesto.org/iso/es/manifesto.html
- Serena Dimensions: http://www.serena.com/index.php/en/products/application-development/dimensions-cm/overview/
- PowerDesigner: http://www.powerdesigner.de/en/
- ClearCase: http://www-03.ibm.com/software/products/es/clearcase
- RAD: http://www-03.ibm.com/software/products/es/application
- Remedy: http://www.bmcsoftware.es/it-solutions/remedy-itsm.html
- Openproj: https://www.openproject.com/
- SVN: https://subversion.apache.org/
- ANT: http://ant.apache.org/
- Hudson CI: http://hudson-ci.org/
- JIRA: https://www.atlassian.com/software/jira/
- Maven: http://maven.apache.org/
- Artifactory: http://www.jfrog.com/artifactory/
- JUnit: http://www.junit.org/
- Selenium: http://seleniumhq.org/
- JMeter: http://jakarta.apache.org/jmeter/
- Sonar: http://www.sonarsource.org/

8. Anexo

En el CD adjunto se incluyen plantillas de documentos generados durante las fases de desarrollo y construcción del software:

Análisis:

- A110 DossierdeUsuario
- A111 - Inventario de Funcionalidades
- A112 Descripción de la Funcionalidad
- C109 Estrategia y Plan de Pruebas.doc
- C111 - Definición de Interfaz

Diseño

- C204 Casos de Prueba
- D110 - Inventario de Componentes
- D111-DetalledelComponente
- F013 - Hoja de Tipificacion Java
- F213 Inventario de Entregas a SwF Java