UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Marco de trabajo para el despliegue de soluciones como servicio utilizando lo esencial de gestión Lean, Kanban, diseño centrado en el usuario y ReactJs

AUTORES: Diego Clerici

DIRECTOR: Patricia Bazán, Elsa Estevez

CODIRECTOR:

ASESOR PROFESIONAL:

CARRERA: Licenciatura en Sistemas – Plan 2015

Resumen

Desarrollo de un marco de trabajo ágil que se basa tanto en la integración de conceptos relacionados tanto a la tecnología informática, como a los conceptos y metodologías para la gestión de proyectos, aplicados a las empresas que ofrecen servicios por internet. Mejora la productividad, velocidad y eficiencia de los equipos, generando valor, reduciendo tiempo y costos.

Palabras Clave

Metodologías ágiles, marco de trabajo, Lean, Lean UX, Lean Startup, Producto Mínimo viable, Diseño centrado en el usuario, Kanban, ReactJs

Conclusiones

Este trabajo promueve el desarrollo de metodologías ágiles para el desarrollo de proyectos orientados a servicios, para empresas con equipos multidisciplinarios. Es muy fácil de implementar, no requiere conocimientos previos en ninguna metodología. Solo entender los pasos para que cada integrante del equipo esté enfocado en lo que sabe hacer. Promueve el pensamiento Lean tanto para el equipo técnico y no técnico. Motiva a "construir el producto correcto" y "construir bien".

Trabajos Realizados

A partir de la introducción de cada uno de los conceptos/términos y/o enfoques a usar por el marco de trabajo propuesto, se definió el mismo y se comparó con otra metodología de desarrollo ágil. Se realizó la implementación de ejemplo de un software utilizando los elementos del marco de trabajo propuesto y se analizaron sus fortalezas y debilidades; en particular, la fluidez de la información y la comunicación dentro de la empresa a partir de la utilización de las metodologías incluidas en el marco de trabajo

Trabajos Futuros

Una línea de trabajo a completar a partir de esta propuesta es, poder plantear un marco flexible para el caso de proyectos para software empresarial y no de consumo. Por ejemplo poder flexibilizar el marco o extenderlo para poder trabajar en un proyecto que considere una etapa de migración de un software viejo a una solución más contemporánea. Otro trabajo a futuro podría ser cómo escalar el proyecto mientras se va desarrollando desde el punto de vista estratégico.

Capítulo 1

Introducción al trabajo de tesis

Introducción

Este trabajo y su propuesta, nace desde la experiencia personal en el ambiente laboral. A lo largo de los años en diferentes empresas en las que me ha tocado trabajar, y los diferentes equipos de los cuales fui parte, me han dado mucho conocimiento tanto tecnológico, técnico, como de gestión de equipos, gestión de proyectos, motivación personal y de equipos, productividad, foco y muchísimas otras herramientas que se necesitan para trabajar en cualquier ambiente laboral.
En particular me he sentido muy a gusto, integrando conceptos relacionados tanto a la tecnología informática, como a los conceptos y metodologías aplicados a las empresas para la gestión de proyectos. Por eso este trabajo, es una mezcla de conceptos muy diferentes, pero que en la práctica conviven muy bien, generando un flujo o un canal por el que se mejora la productividad, velocidad y eficiencia de los equipos, generando valor y gestionando el tiempo de valor agregado[1], sin incrementar los costos y los riesgos.

Motivación

Pensemos directamente en la definición de la palabra Tecnología. Una de las acepciones de esta palabra dice que “designa a los variados medios artificiales mediante los cuales las personas deliberadamente resolvemos nuestros problemas prácticos. El término incluye todos los artefactos y procesos necesarios para la producción de bienes o la **prestación de servicios** de cualquier naturaleza, así como sus principios organizativos o de funcionamiento” [11].
En esta definición podemos notar que habla de “prestación de servicios”. La definición de esta palabra, nos motiva a pensar en que la tecnología entonces es un medio por el que los humanos creamos y damos servicios. Normalmente las personas cada vez que tienen una necesidad, ya sea para realizar una compra, realizar un trámite, buscar información, llegar a un lugar, etc., consumen servicios en internet que le permiten satisfacer esas necesidades. Ya sea en formato de aplicación para un teléfono o una página web, en muchas ocasiones, el software es disponibilizado o provisto a través de un servicio en la nube (SAAS, Software as a Service). Podemos asumir entonces que el desarrollo de este tipo de software va en estar constante crecimiento por la alta demanda de necesidades. Por ello es necesario un marco de trabajo que esté preparado para estos nuevos desafíos para las empresas. Que sea ágil y efectivo,

[1] El tiempo de valor agregado es un tiempo relacionado con la filosofía Lean. Incluye solo aquellas actividades que realizamos en un proyecto por las cuales un cliente estaría dispuesto a pagar.

simple y productivo. Por esto, proponemos considerar el enfoque Lean para startups, que es una metodología propuesta por Eric Ries para desarrollar negocios y productos.
Desde el punto de vista de la construcción del software es claro que el software servirá mejor al usuario si éste está realmente involucrado en la creación de ese software. En este proceso, por un lado, para el usuario no es demasiado relevante una lista de características a agregar al software, pero si tiene sentido cómo se podrán usar esas características en el software desarrollado. Por otro lado, es más útil para un desarrollador escribir modelos de datos, maquetas de interfaces de usuarios (UI) y empezar a construir un catálogo de tareas (backlog o storyboard) para conocer cómo un usuario puede interactuar con la aplicación. Esta información tiene un significado para los usuarios, les permite ofrecer comentarios significativos sobre la creación de la aplicación. En este escenario, resulta mucho más útil para el desarrollo, este último enfoque, que recibir una especificación que ya ha sido aprobada por el usuario, pero que solo aborda el "cómo" y no el "por qué" de la aplicación.
Preguntándole al usuario qué es lo que realmente quiere lograr, se obtendrá una solución mucho más simple y precisa. Por ello es necesario abordar un paradigma centrado en el usuario.
Para organizar al equipo de trabajo, las tareas de los proyectos, la implementación de requisitos funcionales y no funcionales, mejorar la comunicación del equipo, mejorar la productividad y reducir costos en mantenimiento es necesario tener un modelo de desarrollo de software que acompañe. Para ello aparecen las metodologías ágiles como KANBAN[2] [12].
Por último, también se considerará un framework para construir UI basadas en componentes, creado por Facebook, que permite el desarrollo de la solución integrando componentes visuales reutilizables y componentes lógicos reutilizables, que interactúan entre sí para satisfacer las necesidades del usuario. Por eso, proponemos considerar ReactJs [1] [2].

Objetivos

El objetivo es crear un marco de trabajo basado en el desarrollo ágil para empresas de tecnología que dispongan y provean soluciones como servicios a usuarios a través de internet o en la nube. (Software as a Service SAAS). Este marco de trabajo propone aplicar conceptos de metodologías de desarrollo ágil para equipos como Kanban, el paradigma o filosofía de desarrollo de software centrado en el usuario, enfoque Lean para startups, y reactJS como el framework de implementación para construir el software, utilizando su modelo orientado a componentes de Interfaces de usuario (UI).

[2] (letrero o valla publicitaria en japonés), es un sistema de información que controla de modo armónico la fabricación de los productos necesarios en la cantidad y tiempo necesarios en cada uno de los procesos que tienen lugar tanto en el interior de la fábrica, como entre distintas empresas.

Este marco de trabajo promueve la utilización de estos conceptos y tecnologías, que sean capaces de convivir en conjunto para generar procesos de trabajo ágiles. El desafío es seleccionar los elementos esenciales de cada uno de los conceptos y tecnologías para generar un flujo de trabajo ágil.
Para cumplir con el objetivo general arriba descrito, se proponen los siguientes objetivos específicos:

- Introducir cada uno de los conceptos/términos y/o enfoques a usar por el marco de trabajo propuesto.
- Definir el marco de trabajo en función de los elementos seleccionados.
- Comparar con otra metodología de desarrollo ágil.
- Realizar una implementación de ejemplo de un software utilizando los elementos del marco de trabajo propuesto.
- Analizar fortalezas y debilidades del marco de trabajo; en particular, la fluidez de la información y la comunicación dentro de la empresa a partir de la utilización de las metodologías incluidas en el marco de trabajo.

Organización del trabajo de tesis

El trabajo estará organizado de la siguiente manera:

En el capítulo 1 estarán la introducción, la motivación y objetivos de este trabajo.

En el capítulo 2 desarrollaremos lo esencial del pensamiento y gestión Lean y cómo se aplica a la organización de proyectos y a las empresas. También analizaremos la diferencia entre proyectos exitosos y fracasados. Definiremos al manifiesto ágil, y compararemos la gestión tradicional de proyectos con la gestión ágil.

En el capítulo 3 desarrollaremos lo esencial de Kanban y lo principios de Lean development. Hablaremos de cómo se relaciona el pensamiento Lean con esta metodología para la gestión de tareas y proyectos.

En el capítulo 4 desarrollaremos los conceptos de diseño centrado en el usuario, y lo compararemos con modelos convencionales en donde el usuario es relegado a un lugar donde no tiene presencia en las definiciones y validaciones principales del producto o proyecto a desarrollar. Este concepto es fundamental para entender por qué elegimos la tecnología de implementación que desarrollaremos en el capítulo 5.

En el capítulo 5 desarrollaremos los conceptos de ReactJs, en qué está basado, para qué sirve, quién lo creó y para qué, y trataremos de conectar los principios de esta tecnología con una metodología de desarrollo orientada al usuario o basado en el usuario. También compararemos cómo sería implementar una solución utilizando otro framework o ningún framework.

En el capítulo 6 desarrollamos la propuesta del marco de trabajo aplicando todos los conceptos.

En el capítulo 7 utilizaremos un ejemplo real de aplicación del marco de trabajo de manera teórico / práctica para verlo en funcionamiento.

En el capítulo 8 haremos una comparación entre el marco de trabajo propuesto y un marco de trabajo ágil como es SCRUM.

En el capítulo 9 escribiremos las conclusiones y los posibles trabajos a futuro que se podrían desarrollar utilizando este marco como base principal.

Capítulo 2

Gestión Lean y Ágil de proyectos

Introducción

La principal idea de la gestión Lean y ágil de proyectos, es fortalecer nuestras capacidades en dos principales aspectos [12]:

- Acelerar proyectos, sin agregar costos ni reducir la calidad.
- Lograr la eficiencia en la gestión de proyectos, a través de la eliminación de excesos.

Para alcanzar estos objetivos de “velocidad” y “eficiencia” en la dirección de proyectos, abordaremos, a lo largo de este capítulo, éxito y fracaso de proyectos, la evolución de buenas prácticas en la gestión de proyectos a lo largo de los años, el pensamiento Lean y el manifiesto Ágil [17], algo así como la biblia para gestionar proyectos de manera Lean y ágil.

Proyecto exitoso y Fracasos de proyectos

Tanto el fracaso como el éxito es una cuestión relativa, todo depende de la definición o estándar que queramos utilizar para medir un proyecto.
Si bien las técnicas de administración de proyectos se utilizan desde hace varios siglos, el auge y desarrollo de herramientas específicas comenzó a profundizarse a partir de 1960.
En la década de 1960 se definía al éxito de un proyecto solo en función de su **calidad**. O sea, un proyecto que cumpliera con los objetivos de calidad preestablecidos se lo definía como exitoso.
Luego, a partir de la década de 1980, se define a un proyecto exitoso, cuando además de cumplir con la calidad, cumplía con los **plazos** y **presupuestos** según el plan del proyecto.
Como si esto fuera poco, a partir de 1990, no alcanza con cumplir calidad, plazos y presupuesto para el éxito de un proyecto. Sino, que además de estos objetivos mínimos, es necesario que el proyecto cumpla con la “**satisfacción del cliente**”.
¿De qué serviría un proyecto de una calidad excepcional, que se finalizó en el plazo previsto, utilizando los recursos preestablecidos, si luego nadie compra los productos de ese emprendimiento?

A estas cuatro características de proyecto exitoso debemos agregar también la “**sostenibilidad o cuidado**”. O sea, no podríamos definir como exitoso un proyecto que cumplió con parámetros técnicos de calidad, cronograma, presupuestos y

satisfacción del cliente, si no fuimos capaces de preservar el medio ambiente o los miembros del equipo durante la ejecución del proyecto. Por ejemplo, si el proyecto para cumplir con los parámetros técnicos fue tan exigente que todos los miembros del equipo terminaron muy desgastados físicamente y/o peleados entre ellos, seguramente no podremos volver a disponer de estas personas en proyectos similares, por lo que la definición de proyecto exitoso podría verse opacada.
Por ende, hasta el día de la fecha, para que un proyecto sea exitoso debería cumplir con los siguientes requisitos:

- Presupuesto.
- Plazo.
- Calidad.
- Aceptación del cliente.
- Sostenibilidad.

Existen muchos fracasos de proyectos reales que no cumplieron con alguno de los parámetros mencionados previamente. Veamos tan sólo algunos ejemplos.
La Opera House en Sydney[3] es un proyecto del que casi todos los australianos están orgullosos. Sin embargo, desde la definición técnica de éxito, no podemos decir que fue exitoso, ya que según el presupuesto base habían estimado invertir 7 millones de dólares y el proyecto finalizó con una inversión de 107 millones de dólares. Queda claro que ese error de cálculo de 100 millones de dólares lo coloca en la lista de proyectos no exitosos por incumplimiento del presupuesto.
El Eurotunel[4] es uno de los proyectos que no supieron cumplir con el cronograma. En base al plan se debería haber entregado en el año 1992, no obstante, tuvieron una demora de dos años, abriendo las puertas al público en el año 1994. No sólo eso, sino que también tuvieron problemas presupuestarios, ya que el tiempo es dinero. Partiendo de una inversión base estimada en 7.500 millones de dólares, el proyecto finalizó con una inversión de 17.500 millones. Ese desvío de 10.000 millones de dólares fue un costo hundido y bien hundido debajo de toda esa agua, que los inversores no podrán recuperar jamás con la operación del proyecto.
El Tacoma Narrow Suspension Bridge[5], también denominado “Puente Galopante” por las fuertes ondulaciones que tenía cuando soplaba viento por la zona, la perfección tanto con la agenda como con el presupuesto, lo cual respetaron al pie de la letra sin ningún inconveniente. En relación a la calidad, habían planificado que el puente podría soportar vientos de hasta 120 millas por hora. Sin embargo, un día cuando el viento soplaba tan sólo a 40 millas por hora, el puente comenzó a sufrir fuertes ondulaciones que lo destruyeron por completo. Esta catástrofe ocurrió a los 4 meses después de su inauguración oficial en el año 1940, lo que dio fin al tercer puente más

[3] https://www.sydneyoperahouse.com/
[4] https://es.wikipedia.org/wiki/Eurot%C3%BAnel
[5] https://en.wikipedia.org/wiki/Tacoma_Narrows_Bridge_(1940)

largo del mundo de esa época. ¿De qué nos sirve preocuparnos tanto por los presupuestos y el cronograma, si descuidamos la calidad?
El Concorde[6] fue un avión diseñado para que volara más rápido que el sonido. La tecnología utilizada fue admirable, el avión unía la ruta Londres – Nueva York en un poco menos de 4 horas. Existen tantos negocios entre esas dos ciudades que todos pensaban que sería un gran éxito comercial. Por ejemplo, un empresario que viviera en una de esas ciudades, podría trasladarse hacia la otra para una reunión presencial y volver a dormir a su casa en el mismo día. Sin embargo, el proyecto nunca alcanzó "satisfacción del cliente". Para que el negocio cubriera su punto de equilibrio de costos operativos, el pasaje de la ruta mencionada tenía que ser vendido a 8.000 dólares como mínimo y el avión debería estar lleno de pasajeros. Al parecer, nunca encontraron tantas personas con ganas de hacer negocios por un valor tan caro del pasaje y en octubre del año 2003 el proyecto cerró sus puertas.
Durante el proyecto de simulacro de corte de energía en Chernóbil[7] en 1986, ocurrió uno de los desastres ambientales más grandes de la historia. La explosión de la planta nuclear mató a 33 personas y contaminó como mínimo a otras 600.000 con radiaciones nucleares. Hasta el día de la fecha no están claras las muertes posteriores que ha ocasionado ese gran fracaso de proyecto. Aunque ese proyecto de corte de planta hubiera estado dentro del presupuesto y el cronograma, queda claro que no fue demasiado exitoso.
En todos estos ejemplos de fracasos de proyecto, queda en evidencia la necesidad de tener cumplir con todos los requisitos planteados en la defínción de proyecto exitoso.

Evolución de buenas prácticas

A los fines de mejorar la eficiencia en los proyectos para estar más cerca de los exitosos, que de aquellos que fracasan, el mundo ha tenido una gran evolución de buenas prácticas.

Gráfico 1.1 - Evolución de modelos de gestión

Si tomamos como referencia los últimos 100 años, el primer cambio drástico en la forma de gestionar proyectos comenzó en la década de 1920 con las enseñanzas de Henry Ford y sus modelo de **producción masiva**.

[6] https://es.wikipedia.org/wiki/Concorde
[7] https://es.wikipedia.org/wiki/Accidente_de_Chern%C3%B3bil

Durante la era Ford mejoró muchísimo la eficiencia y productividad, al ordenar los procesos productivos con la especialización y división del trabajo, para que cada departamento de la empresa se especializa en lo que mejor sabía hacer. Estos conceptos ya habían sido pregonados previamente por el Economista estadounidense Frederick Taylor[8].

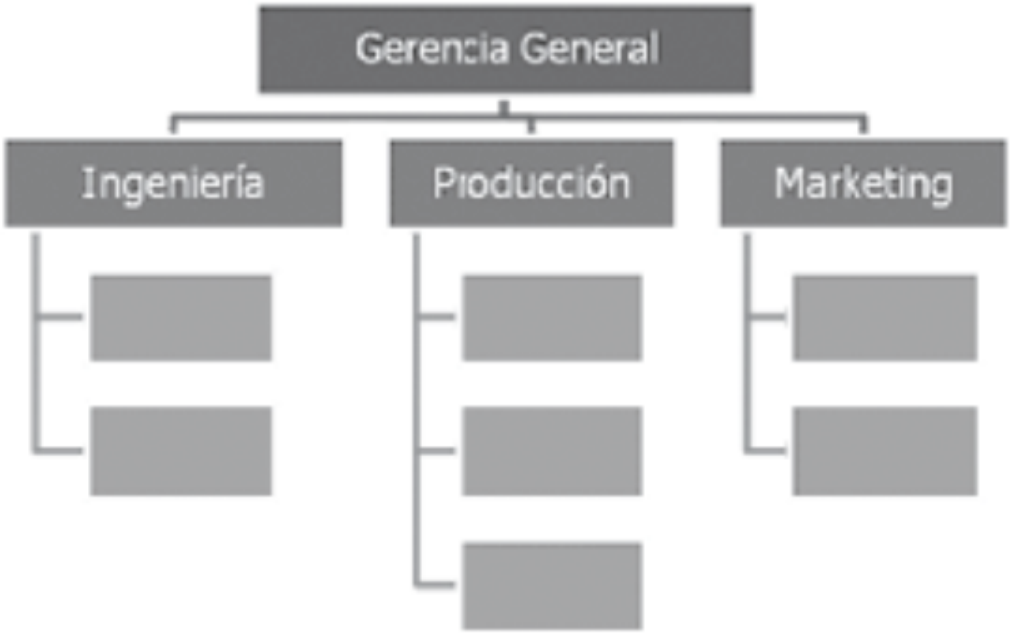


Gráfico 1.2 - Producción masiva, departamentos funcionales independientes.

El otro cambio ocurre aproximadamente en 1960, cuando se empieza a prestar mucha más atención a los procesos para la gestión de calidad en los proyectos. Había llegado la era de la calidad total al momento de gestionar cualquier tipo de proyecto relacionado con la producción masiva.
Y estos procesos vinieron de la mano de una organización matricial, en lugar de departamentos funcionales independientes. Algunas empresas comenzaron a crear un nuevo departamento funcional denominado PMO (Project Management Office), o por lo menos, nombrar a Directores de Proyectos con la autoridad suficiente para utilizar recursos de los distintos departamentos de la compañía al momento de la ejecución de un proyecto.

[8] Frederick Winslow Taylor (20 de marzo de 1856 - 21 de marzo de 1915) fue un ingeniero Industrial y economista estadounidense, promotor de la organización científica del trabajo y es considerado el padre de la Administración Científica.

Gráfico 1.3 - Organización matricial. PMO

A partir de 1985, Motorola pone de moda el concepto de “Six Sigma”, donde las empresas tenían que seguir mejorando la eficiencia en sus procesos de gestión de calidad. Por ejemplo, si la empresa se dedicaba a la producción de bienes o servicios, solamente podía tener 4 fallas cada un millón de bienes o servicios que salían al mercado.

En esta época se empieza a proponer la “Ingeniería concurrente” para que no solamente trabajara el Director de Proyectos con distintas miembros de la organización de manera matricial, sino que también era muy conveniente incluir en las grandes decisiones sobre el proyecto a la alta gerencia, para evitar los constantes cambios que sufrían los proyectos.

Un hito importante de esta era se da cuando el Project Management Institute[9], la organización sin fines de lucro sobre dirección de proyectos mas grande del mundo, lanza al mercado la primera edición del PMBOK[10] , donde se deja por escrito que para obtener un proyecto exitoso no alcanza con enfocarse solamente en el área de calidad. Por el contrario, hay que planificar y gestionar los proyectos de manera integral incluyendo procesos en las siguientes áreas del conocimiento:

[9] El Project Management Institute (PMI) es una organización estadounidense sin fines de lucro que asocia a profesionales relacionados con la Gestión de Proyectos. Desde principios de 2011, es la más grande del mundo en su rubro, dado que se encuentra integrada por cerca de 500 000 miembros en casi 100 países. La oficina central se encuentra en la localidad de Newtown Square, en la periferia de la ciudad de Filadelfia, en Pensilvania (Estados Unidos).

[10] La Guía de los fundamentos para la dirección de proyectos (del inglés A Guide to the Project Management Body of Knowledge o PMBOK por sus siglas) es un libro en el que se presentan estándares, pautas y normas para la gestión de proyectos. La última versión publicada es la 6ª, publicada el 6 de septiembre de 2017.

- Alcance
- Tiempo
- Costo
- Calidad
- Recursos humanos
- Comunicaciones
- Riesgos
- Adquisiciones
- Integración

Aproximadamente en el año 1995 se pone en auge el concepto de **“Lean”** debido a los excelentes procesos que estaba implementando Toyota en sus proyectos. Estos conceptos son replicados rápidamente en el resto del mundo y las empresas comienzan a mejorar la eficiencia en sus procesos.
El mundo reconoce que era importante trabajar los proyectos con procesos formales, pero si estos procesos estaban demorando demasiado los proyectos, era necesario pulir o eliminar aquellos que no eran necesarios.
Comienzan a desarrollarse proyectos donde tenían que tener en cuenta un intercambio inteligente entre “procesos y control de gestión” vs. “velocidad y valor al cliente”. Se busca mantener un **flujo continuo** de valor al cliente sin interrupciones.
Si bien todas las buenas prácticas que se habían desarrollado hasta el momento eran excelentes para la producción masiva, algunas veces se estaba pecando en exceso de procesos, lo que ponía lentos a los proyectos y descuidaba al cliente.
Los conceptos Lean llegan para remover todos esos excesos de programación, para poder enfocarse en llevar valor al cliente lo más rápido posible.
En el año 2001 un grupo de ingenieros informáticos de Utah, Estados Unidos, escribe el **Manifiesto Ágil** para la gestión de proyectos de software.
Estos conceptos rápidamente se hacen populares en el mundo entero y comienzan a ser utilizados no sólo para el desarrollo de software, sino para cualquier otro tipo de proyectos.

Gráfico 1.4 - Evolución de los conceptos de buenas prácticas.

Qué es Lean

Lean es un amplio eslogan que describe un enfoque holístico y sostenible que usa menos de todo para brindarte más. Lean es una estrategia de negocios basada en satisfacer al cliente mediante la entrega de productos y servicios de calidad que son justo lo que el cliente necesita, cuando el cliente los necesita, en la cantidad requerida, al precio correcto, mientras utiliza el mínimo de materiales, equipo y espacio, trabajo y tiempo. Las prácticas Lean permiten a una organización reducir sus ciclos de desarrollo, producir productos y servicios de mayor calidad a costos más bajos y usar los recursos de manera más eficiente [12] [13] [18] [19].
Lean significa menos de muchas cosas: menos desperdicio, ciclos más cortos, menos proveedores, menos burocracia. Pero Lean también significa más: más conocimiento y empoderamiento de los empleados, más agilidad y capacidad organizativa, más productividad, más clientes satisfechos y más éxito a largo plazo.

Clasificación de Lean

Lean es una iniciativa de mejora empresarial aplicada en toda la empresa. Un error común de interpretación sostiene que Lean es una especie de programa de calidad solo para la fabricación de productos físicos. No es tan así. La filosofía, los principios y las prácticas de Lean son aplicables en cualquier lugar y son más efectivos cuando se aplican en toda la organización. Existe una clasificación de distintos grupos que utilizan Lean [18] [19]:

- **Lean Production o Lean Manufacturing:** Al principio de la formalización de las técnicas Lean, las prácticas se modelaron según los enfoques de fabricación y producción en compañías como Toyota. En otras empresas de fabricación se obtuvieron enormes éxitos, ya que los profesionales de Lean aplicaron las técnicas en otros entornos de fabricación.

- **Lean Office y Lean Administration:** Estas referencias señalan que las prácticas se han aplicado con gran éxito en los entornos de oficina, donde las corrientes de valor se basan en políticas, en la toma de decisiones orientada a la información e implican la gestión efectiva de transacciones y datos.

- **Lean Management:** Este término suele asociarse con el rol de los gerentes en la empresa Lean. Esto cubre la gestión de una iniciativa Lean, así como las prácticas Lean personales de los propios gerentes.

- **Lean Startup:** De manera similar a los preceptos del lean management, la filosofía de lean startup de Eric Ries busca eliminar las prácticas ineficientes y se centra en incrementar el valor de la producción durante la fase de desarrollo. De esta forma la startup puede tener más oportunidades de triunfar sin requerir

grandes cantidades de fondos externos, planes de empresa elaborados, o el producto perfecto.

- **Lean Thinking:** Debido a que Lean es más que solo herramientas y técnicas, las personas dentro de una organización Lean efectiva aplican las prácticas Lean como una forma de pensar, una forma de abordar problemas y desafíos. "*Una vez que haya adoptado verdaderamente las formas de Lean, será un pensador Lean.*" Lean Thinking[11] también es el nombre de un libro de James Womack y Daniel Jones, publicado por primera vez en 1996, que marca un hito en Lean. Fue en este trabajo histórico que todos empezaron a asociar a Lean con algo más que Toyota, y comenzaron a pensar en Lean como un movimiento propio.

De estos grupos o clasificaciones dónde se aplica Lean, en este trabajo nos enfocaremos en Pensamiento Lean (Lean Thinking), y algunos conceptos de Lean Startup. Más adelante veremos cómo esta forma de pensar Lean, nos ayudará a tomar decisiones en todos los contextos y en cualquier momento de un proyecto.

Lean Startup

Vivimos en una era de oportunidades sin precedentes para la innovación. Con el advenimiento de Internet, la computación en la nube y el software de código abierto, el costo de los productos de construcción está en un mínimo histórico. Sin embargo, las probabilidades de construir nuevas empresas exitosas no han mejorado mucho.
La mayoría de las startups todavía fallan.
Pero el hecho más interesante es que, de esas nuevas empresas que tienen éxito, dos tercios informan haber cambiado drásticamente sus planes a lo largo del camino.
Entonces, lo que separa a las startups exitosas de las que no tienen éxito no es necesariamente el hecho de que las startups exitosas comenzaron con un mejor plan inicial (o Plan A), sino que encuentran un plan que funciona antes de quedarse sin recursos.
Hasta ahora, encontrar este mejor Plan B o C o Z se ha basado más en el instinto, la intuición y la suerte. No ha habido un proceso sistemático para realizar pruebas rigurosas de estrés en un Plan A.
De eso se trata Running Lean [19].
Ejecutar Lean es un proceso sistemático para iterar desde el Plan A hasta un plan que funcione, antes de quedarse sin recursos.
Lean Startup es un término registrado por Eric Ries y representa una síntesis de las metodologías de desarrollo de clientes, desarrollo de software ágil y prácticas Lean (como en el sistema de producción de Toyota).

[11] https://books.google.com.ar/books/about/Lean_thinking.html?id=aGEqywqkk00C&source=kp_book_description&redir_esc=y

El término Lean es a menudo mal entendido como "ser barato". Mientras que "ser Lean" se trata fundamentalmente de eliminar el desperdicio o ser eficiente con los recursos, esa interpretación no es completamente errónea porque el dinero es uno de esos recursos.
Ries utiliza, tanto en su blog como en su libro, una terminología específica para describir los principios esenciales del lean startup [21].

Producto mínimo viable

Un producto mínimo viable (PMV o Minumum Viable Product MVP en Inglés) es “la versión de un nuevo producto que permite a un equipo recoger con el mínimo esfuerzo la máxima cantidad de conocimiento validado acerca de los consumidores”. El objetivo de un PMV es evaluar las hipótesis fundamentales de un negocio (o “actos de fe”) y ayudar a los emprendedores a comenzar el proceso de aprendizaje lo más rápido posible. Como ejemplo, Ries destaca que el fundador de Zappos[12], Nick Swinmurn, quería probar la hipótesis de que los clientes estaban dispuestos y querían comprar zapatos en línea. En lugar de desarrollar una página web y una enorme base de datos sobre calzado, Swinmurn se puso en contacto con zapaterías locales, tomó fotografías de su inventario, subió las imágenes a Internet, les compró los zapatos al precio de mercado, y los vendió directamente a los clientes si los compraban a través de su sitio web. Swinmurn dedujo que existía una demanda latente por parte de los consumidores, y Zappos se convirtió en un negocio multimillonario basando su estrategia en la venta de zapatos en línea [20] [21].

Producción continua para desarrollo de software

La puesta en producción continua (Continuous deployment) es un proceso “donde todo el código que se escribe para una aplicación se pone en producción de forma inmediata”, lo que redunda en una reducción de los ciclos de entrega del producto. Ries destaca que en alguna de las compañías para las que ha trabajado, el código se va poniendo en producción hasta unas cincuenta veces al día. Este concepto fue creado por Timothy Fitz, uno de los compañeros de Ries y de los primeros ingenieros en trabajar en IMVU[13] [21].

Experimentos split-test

Un experimento split-test o experimento A/B es aquél en el que se ofrecen “diferentes versiones de un producto al mismo tiempo”. El objetivo de un experimento split-test es observar los cambios en el comportamiento entre los dos grupos para medir el impacto de cada versión en un indicador accionable.
Los experimentos A/B también se pueden realizar en serie, de tal forma que un grupo de usuarios una semana puede ver una versión del producto mientras que, a la

[12] https://www.zappos.com/

[13] IMVU es un juego social con un entorno metaverso y de mensajería instantánea accesible a nivel mundial. Es fundado en el 15 de abril de 2004 en EEUU encabezado por Eric Ries, Will Harvey, Matt Danzig y cuentan con un director ejecutivo llamado Brett G. Durrett.

siguiente, ven otra distinta. Esta forma de trabajar puede plantear dudas en las circunstancias donde eventos externos pueden influenciar el comportamiento en un período, pero no en otro. Por ejemplo, un split-test de dos sabores de helado realizados en serie durante el verano y el invierno podría mostrar una bajada acentuada en la demanda durante el invierno, debido al tiempo atmosférico, no al propio sabor ofrecido [21].

Indicadores accionables

Los indicadores accionables permiten tomar decisiones de negocio con criterio y establecer las acciones que sean pertinentes. Por el contrario, los indicadores "vanidosos" ofrecen mediciones sesgadas, mostrando el mundo "de color de rosa", pero no reflejan de forma adecuada los auténticos motores de crecimiento de una empresa.

Los indicadores vanidosos de una empresa pueden ser accionables para otra. Por ejemplo, una compañía especializada en crear cuadros de mando para los mercados financieros podría estar utilizando la cifra de páginas vistas por persona como un indicador vanidoso, dado que sus ingresos no están basados en esa métrica. Sin embargo, una revista en línea que muestra publicidad puede ver el número de páginas vistas como un indicador esencial, puesto que está directamente correlacionado con la cifra de ingresos de su negocio.

Un ejemplo típico de indicador vanidoso es "el número de usuarios nuevos por día". Aunque un número alto de usuarios nuevos al día parezca beneficioso para cualquier empresa, si el precio de adquirir cada usuario a través de costosas campañas de publicidad es significativamente más alto que los ingresos que se generan por usuario, entonces aumentar su número podría conducir rápidamente a la bancarrota [21].

Pivote

Un pivote es una "corrección estructurada diseñada para probar una nueva hipótesis básica sobre el producto, la estrategia y el motor de crecimiento". Un ejemplo destacable de una empresa que utiliza el pivote es Groupon[14]; en sus comienzos, era una plataforma de activismo llamada The Point.[15] Ante la falta de repercusión, los fundadores crearon un blog en WordPress y lanzaron su primer cupón promocional para utilizarlo en la pizzería situada en la recepción de su edificio de oficinas. A pesar de que sólo se canjearon 20 cupones, los fundadores se dieron cuenta de que su idea era importante, y había llevado a la gente a coordinar una acción de grupo. Tres años más tarde, Groupon creció hasta convertirse en un negocio multimillonario [21].

[14] https://es.wikipedia.org/wiki/Groupon

[15] Penenberg, Adam. Eric Ries Is A Lean Startup Machine. Fast Company. September 8, 2011.

La contabilidad de la innovación

Este tema se refiere a cómo los emprendedores pueden mantener su responsabilidad y maximizar los resultados, midiendo el progreso, planificando hitos, y priorizando tareas [21].

Crear-Medir-Aprender

El circuito Crear-Medir-Aprender es el núcleo central de la metodología lean startup y explica lo que se debería hacer entre las fases de ideación (Crear), codificación (Medir) y verificación de datos (Aprender). En otras palabras, es un proceso iterativo de transformar ideas en productos, medir la reacción y comportamiento de los clientes frente a los productos y aprender si perseverar o pivotar de idea. Este proceso se repite de forma continuada [21].

Lean Thinking

Para entender el pensamiento Lean es importante entender cómo fue descubierta en 1988 por un grupo de investigadores del MIT (Massachusetts Institute of Technology), dirigidos por el Dr. James P. Womack[16]. Ellos investigaban sobre la industria automotriz internacional, y observaron comportamientos únicos en Toyota.
El investigador John Krafcik[17] y los otros lucharon con un término para describir lo que estaban viendo. Ellos observaban los atributos de rendimiento del sistema de Toyota, comparado con la forma tradicional de producción. Lo que observaron de Toyota fue [12]:

- Necesitaban menos esfuerzo para diseñar, fabricar y dar servicio a sus productos.
- Necesitaban menos inversión para alcanzar un determinado nivel de capacidad de producción.
- Producían productos con menos defectos.
- Utilizaban menos proveedores.
- Ejecutaban sus procesos principales en menos tiempo y esfuerzo.
- Necesitaban menos inventario en cada paso.
- Tenían menos empleados accidentados.

[16] James P. Womack fue el director de investigación del Programa Internacional de Vehículos Motorizados (IMVP) en el Instituto de Tecnología de Massachusetts (MIT) en Cambridge, Massachusetts, y es el fundador y presidente de Lean Enterprise Institute, una institución sin fines de lucro para la difusión y exploración. del pensamiento Lean con el objetivo de su posterior desarrollo de Lean Enterprise.

[17] John Krafcik (nacido el 18 de septiembre de 1961) es el CEO de Waymo. Krafcik fue el ex presidente de True Car Inc. y el presidente y director ejecutivo de Hyundai Motor America. Fue nombrado director general del proyecto de autos de auto conducción de Google en septiembre de 2015. Krafcik se mantuvo como director general después de que Google separó su proyecto de auto-manejo de automóviles y lo convirtió en una nueva compañía llamada Waymo, ubicada bajo la compañía matriz de Google, Alphabet Inc.

Ellos concluyeron que una compañía como esta, una compañía que usa menos de todo, es una compañía “Lean”.
Y así como así, el término Lean se asoció con una cierta capacidad de negocio: la capacidad de lograr más con menos. Menos esfuerzo humano para realizar su trabajo, menos material para crear sus productos y servicios, menos tiempo para desarrollarlos, menos energía y espacio para producirlos. Ellos están orientados directamente hacia la demanda del cliente, y desarrollan productos y servicios de la manera más efectiva y económica posible.

La filosofía Lean se puede resumir en cinco principios fundamentales [12]:

1. Especificar con precisión el **valor** de cada proyecto
   Se considera **“Valor”** a cualquier cosa por la que un cliente estará dispuesto a pagar. Las actividades sin valor son consideradas desperdicios.
2. Identificar el **flujo de valor** del proyecto.
   El **flujo de valor** se compone de todas las tareas necesarias que deben ser completadas para entregar el producto o servicio al cliente. Si nos enfocamos en los entregables del proyecto, estaremos construyendo el flujo de valor.
3. Permitir que el valor fluya **sin interrupciones**
   El valor tiene que llegar al cliente lo más rápido posible. Hay que identificar los tiempos de demora en el flujo y **quitar los obstáculos innecesarios** en el proceso.
4. Permitir que el cliente participe en la identificación de valor.
   Los equipos de proyectos deberían permitir a sus clientes que los ayuden a **identificar lo que agrega valor.**
5. Buscar de manera continua la perfección.
   Una vez introducida la cultura Lean a los proyectos, es necesario **seguir mejorando de manera continua.**

Sin embargo, en un Lean Startup, nos esforzamos por optimizar la utilización de nuestro recurso más escaso, que es el tiempo. Específicamente, nuestro objetivo es maximizar el aprendizaje (sobre los clientes) por unidad de tiempo.
La conclusión clave de Lean Startup se puede resumir mejor en torno al concepto de usar iteraciones más pequeñas y rápidas para probar una visión.

Manifiesto Ágil

En febrero del año 2001 un grupo de ingenieros informáticos se reunieron en Utah para redactar el Manifiesto Ágil, ya que enfrentaban demasiados problemas cuando querían gestionar proyectos informáticos con las prácticas tradicionales que existían hasta ese momento. Y de manera simplificada escribieron lo siguiente [17]:

“Estamos descubriendo formas mejores de desarrollar software tanto por nuestra propia experiencia como ayudando a terceros. A través de este trabajo hemos aprendido a valorar:
***Individuos e interacciones** sobre procesos y herramientas.*
***Software funcionando** sobre documentación extensiva.*
***Colaboración con el cliente** sobre negociación contractual.*
***Respuesta ante el cambio** sobre seguir un plan.*
Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.”

También se definieron 12 principios en los que se basa el desarrollo ágil [17]:

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.

Estos principios son bastantes similares a la filosofía Lean. Aparecen conceptos como cliente, valor, flujo continuo y mejora continua.
En la actualidad esta corriente del pensamiento ágil es más popular que la filosofía Lean, pero en varios puntos podríamos decir que tienen similitudes.

Detrás del pensamiento ágil existen varias herramientas o técnicas específicas para gestionar proyectos de software como por ejemplo [12]:

- Crystal
- Extreme Programming.
- Feature-Driven Development
- Kanban
- Scrum

Para este trabajo, seleccionamos a Kanban como herramienta para la gestión ágil de las tareas del proyecto. Abordaremos más en detalle este tema en el capítulo 3.

Gestión tradicional vs Ágil

En la gestión tradicional de proyectos se suelen encontrar algunas de las siguientes características [12]:

- La elaboración de propuestas está llena de desperdicios.
- Existen demoras innecesarias debido a baches en el flujo de valor.
- Las reuniones de coordinacion y colaboracion insumen más tiempo del necesario
- En algunos casos extremos se llevan a cabo reuniones eternas
- Las empresas avalan y premian la gestión de héroes y súper-genios, dueños de la información.
- Las estimaciones de tiempos, costos y calidad se basan en son imprecisas.
- No se tienen en cuenta la planificacion y administracion de riesgos.
- En el mejor de los casos, se utiliza solo una planificación en base a la ruta crítica.
- Suelen existir procesos burocráticos con esquema PODeL (por orden de llegada)
- Es normal y habitual trabajar en un ambiente de multi tareas simultáneas.
- Se requiere un alto grado de desgaste para alcanzar los objetivos.

Por el contrario, en un esquema de gestión ágil de proyectos, algunas características suelen ser [12]:

- La elaboración de propuestas se concentra en la necesidad del cliente sin agregar desperdicios adicionales.
- Se identifica el flujo del valor del proyecto y se crea un ambiente ágil que permita que fluya el valor sin interrupciones.
- Los equipos están comprometidos con el proyecto y los dueños de la información no son bien vistos por la organización.
- Se trabaja en base a procesos.

- Se tienen en cuenta la planificación y administración de riesgos en los proyectos.
- Se amplía la visión de la ruta crítica para incluir los recursos críticos asociados a cada actividad.
- Se trabaja con un esquema donde se planifican reservas, con un sistema de turnos para evitar colas y esperas.
- Se planifican y priorizan todas las actividades de los programas o proyectos, a fin de evitar la realización de multitareas en forma simultánea.
- Se alcanzan buenos resultados con mayor calidad de vida, en relación a la gestión tradicional.

Conclusiones

A lo largo de este capítulo pudimos recorrer en el tiempo, cómo fue cambiando la gestión de proyectos en base a las necesidades y el aprendizaje que se iba adquiriendo con los años. El objetivo principal es lograr proyectos exitosos. Por eso se fueron cambiando los esquemas, los roles, las estructuras internas de las empresas para lograr diferentes modelos de funcionamiento y así lograr el éxito. Se pasó por un modelo de producción masiva, enfocado en la productividad y eficiencia. Luego a un modelo de calidad total donde el foco estaba puesto en la producción con calidad y la aparición del PMO, dentro de la estructura de la empresa. Luego vino SIx Sigma, introducido por Motorola, junto con la creación del PMBOK.
Por último aparece Toyota con su gestión ágil, pensamiento Lean y Kanban.
Vimos que el enfoque Lean/Agil, termina siendo más beneficioso tanto para la empresa, como para los clientes. Donde se plantea que el cliente es la parte importante de los proyectos. Tanto así que se define el concepto de valor, como aquel entregable que el cliente estaría dispuesto a pagar.
Por último el desarrollo del manifiesto ágil, por ingenieros informáticos, pone en evidencia la relación entre el pensamiento o filosofía Lean y la gestión ágil de proyectos de software.

Capítulo 3

Kanban

Introducción

Kanban significa señal o letrero en japonés y es el nombre que se le da al mecanismo que se implementa a través de una tarjeta, la cual va fluyendo desde la recepción de la orden, hasta que el trabajo se encuentra terminado.
Kanban es utilizado como parte central de Lean, que surge como un paradigma de producción introducido en los años 90 en Japón por Toyota como contraposición a los esquemas de producción masiva de los grandes productores norteamericanos. El desafío consistía en lograr bajos costos de producción, ya que por esos tiempos la gente en Japón no disponía de muchos recursos y además el país contaba con un mercado mucho menor que el de las empresas americanas. Estas últimas se basaban en en líneas de producción masivas, que al fabricar automóviles para un mercado importante en volumen, permitían bajar considerablemente los costos. El desafío en Toyota era lograr costos bajos, con volúmenes mucho más reducidos.
Como producto de esta necesidad emerge el Sistema de Producción Toyota, ideado por Taiichi Ohno[18].
La industria Japonesa, a través de la introducción de innovadoras maneras de producción, lograron una reducción del 50% en los esfuerzos de ingeniería y lograron reducir el tiempo de desarrollo en un tercio comparándolo con los enfoques tradicionales. Esos resultados demostraron que introducir cambios en las fases finales de producción eran mejores que los cambios realizados durante la etapa de diseño.
Las empresas americanas, para reducir costos, desarrollaron sus procesos de producción en serie, con muy poca participación de los proveedores durante el mismo. Como resultado de ello se alargaban los procesos de producción debido a la necesidad de incorporar las necesidades de los usuarios tempranamente tal que pudieran ser incorporados al inicio del proceso. En cambio, las empresas japonesas privilegiaron el desarrollo rápido, el acortamiento del ciclo de producción, dejando las decisiones de diseño para las últimas fases de desarrollo.

“No decida que desarrollar hasta tanto no tenga una orden, una vez que la tenga desarrolle el producto lo más rápido que sea posible”

El paradigma seguido por las empresas japonesas fue Lean Development y fue adoptado por varias compañías alrededor del mundo en la década de los 90’.

[18] Taiichi Ohno (Dalian, 29 de febrero de 1912, Toyota City, 28 de mayo de 1990) fue un ingeniero industrial japonés. Es conocido por el sistema de producción Toyota, Just In Time (JIT), dentro del sistema de producción del fabricante de automóviles.

Principios de Lean Development

Los principios seguidos por Lean Development son los siguientes:

- **Eliminación de desperdicios**: Todo aquello que no agregue valor es considerado un desperdicio
- **Amplificar el aprendizaje**: Generar ciclos cortos, que dejen aprendizaje para la toma de decisiones y mejora del ciclo.
- **Tomar decisiones de diseño lo más tarde posible:** Desarrollo concurrente por sobre desarrollo secuencial
- Realizar las entregas lo más rápido posible: Reducir el trabajo en progreso, entregar rápido reduciendo el riesgo. Implementación de un sistema pull.
- **Empoderar al equipo**: Lograr un equipo motivado, liderazgo por sobre gerenciamiento controlado.
- **Desarrollar un producto**: sólido e íntegro. Balanceo entre funcionalidad, usabilidad, confiabilidad y economía que logre satisfacer a los usuarios.
- **Ver el todo**: Visión sistémica del proceso de desarrollo con sistemas dinámicos y el uso de mediciones.

Dentro de lo que Lean propone como “sistema pull” y bajo la utilización de los conceptos de “justo a tiempo”, surge el mecanismo denominado Kanban. El mismo sirve de base para coordinar las tareas de los operarios de la empresa y todo el mecanismo de relacionamiento con los proveedores de las partes que son necesarias para elaborar los productos solicitados.

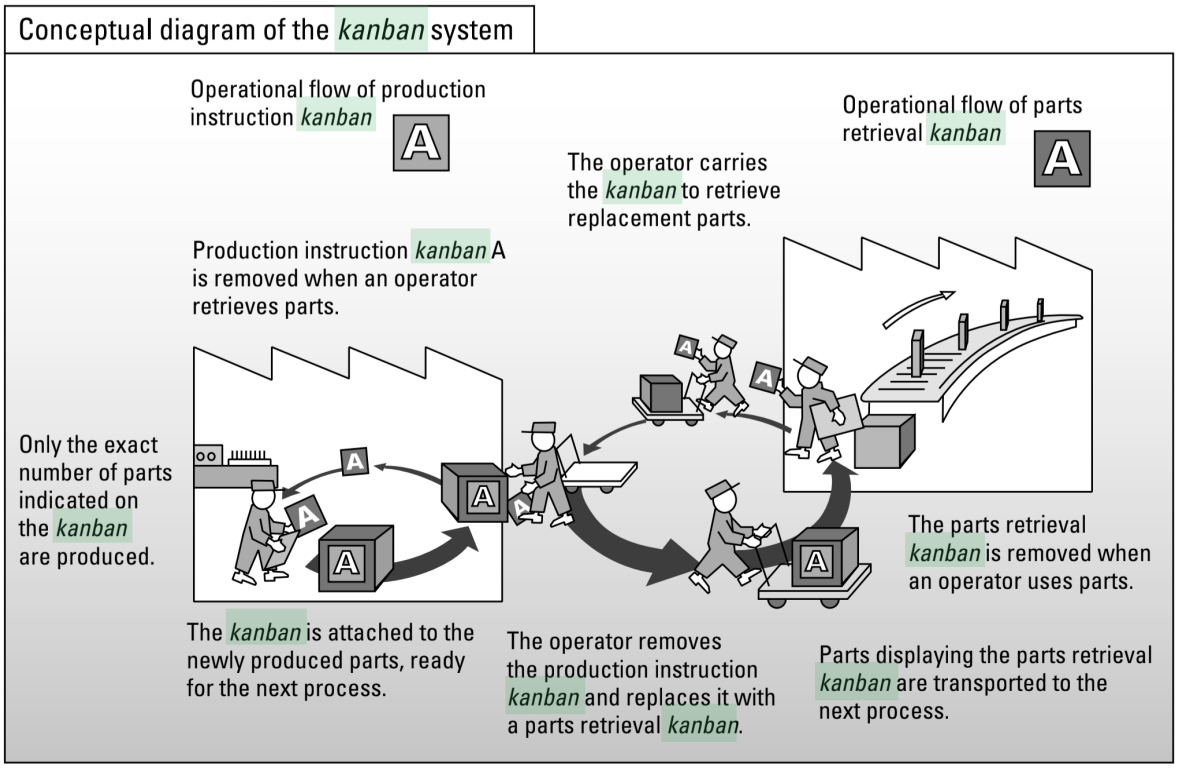


Gráfico 3.1 - Diagrama conceptual del sistema Kanban [18].

A continuación veremos cómo estos conceptos de producción de productos son llevados al desarrollo de software.
Kanban en el desarrollo de software consiste en patrones que son utilizados para analizar un proceso de producción como los implementados por las automotrices japonesas en las últimas décadas.
La utilización de las “Kanban” en el desarrollo de software se implementa a través de una pizarra que visualiza las “órdenes de trabajo” que deben ser procesadas.

Por realizar	Haciendo	Terminado
Tarea A Tarea B	Tarea C Tarea D Tarea E	Tarea F Tarea G Tarea H

La pizarra implementa la lista de solicitudes “A realizar” (backlog) que deben ser procesadas. Estas características son similares a las descriptas para Scrum[19], donde

[19] **Scrum** es el nombre con el que se denomina a los marcos de desarrollo ágiles caracterizados por: Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.

las solicitudes son implementadas en tarjetas, administradas a través de una pizarra, desde el cual los integrantes del equipo “retiran” (pull) las que van a ser procesadas y las trabajan hasta su finalización.

Procesos de desarrollo

El proceso de desarrollo puede ser visto como un esquema de colas (listas de trabajo) las cuales van almacenando las órdenes de trabajo, entre las diferentes etapas del ciclo de desarrollo. Cuando estas colas de trabajo comienzan a crecer, frenando las tareas por diferentes motivos, los tiempos del ciclo de desarrollo comienzan a estirarse y la finalización de las solicitudes se atrasa. Los problemas pueden ser variados, puede haber una sobrecarga de trabajo en una etapa del ciclo, por ejemplo durante el análisis, puede necesitarse finalizar “partes” del producto para continuar con el desarrollo, como sucede con los casos de prueba de una solicitud para poder comenzar con el testeo. Como vimos anteriormente, uno de los principios de Lean es el de reducir el ciclo de trabajo. En este contexto, el desafío se traduce en cómo hacer para que esas “colas” intermedias de órdenes sean reducidas al mínimo y que la operatoria fluya dinámicamente, sin frenarse a lo largo del ciclo de desarrollo. Para saber en qué medida se está logrando este objetivo, Lean y Kanban proponen el uso de una métrica específica denominada Tiempo de Ciclo (o Cicle Time en Inglés). El desafío, queda claro, es lograr el menor tiempo posible de este indicador.
Siguiendo la teoría de colas, existen dos aspectos a ser analizados para medir cuán eficiente se transforma el flujo de trabajo. Uno es medir los arribos (a cada cola, cuántas unidades llegan por unidad de tiempo) y otro es medir el tiempo de servicio (cuántas unidades se procesan por unidad de tiempo). Una forma de contrarrestar un ratio importante de arribos es empaquetar pequeños grupos de órdenes para ser entregadas rápidamente. De esa forma la cola de entrada no llega a acumular demasiadas peticiones, casos contrario, si se tarda mucho tiempo en el procesamiento (si los lotes son muy grandes) la cola toma mayores dimensiones. El otro aspecto a controlar es el tiempo de servicio, en nuestro caso de desarrollo de software, serán las actividades propias del desarrollo. Si queremos acelerar el ciclo de desarrollo, el tiempo de cada instancia del proceso será crucial. Cualquiera de esos “procesadores” que trabaje a menor nivel de respuesta que los otros, comenzará a provocar un incremento en su cola de entrada, que hará mas lento el proceso de desarrollo. El ciclo puede ser visto como un flujo de trabajo, en el que cada etapa del mismo agrega valor al producto ingresado (solicitud), hasta que el mismo se transforma en un producto final que agrega valor al usuario (software funcionando).

Marco de trabajo

El marco de trabajo propuesto por Kanban posee las siguientes características:

- Un tablero que permita visualizar el flujo de trabajo de manera clara y completa.

- El trabajo es dividido en bloques, que son una especie de etapas por las que va pasando una solicitud desde su inicio hasta su completitud.

Utilizando el tablero se intentan controlar dos aspectos fundamentales del proceso:

- El trabajo en progreso, al cual se le asignan valores concretos de cuánto trabajo puede estar en progreso en cada estado del ciclo de desarrollo. Esta restricción permite que cada estación de trabajo se concentre en un número finito y bajo de actividades concurrentes, lo que conduce a trabajar rápidamente con los lotes pequeños, reducir las pérdidas de tiempo por el procesamiento paralelo y lograr ciclos de desarrollo más cortos.
- El tiempo medio que lleva al proceso para completar un elemento.

Beneficios de Kanban

¿Qué aporta Kanban a un proceso de desarrollo de software?

- Visualizar todo el flujo de trabajo que se encuentra en progreso.
- Identificar los “cuellos de botella”, aquellas actividades que están frenando la elaboración de los productos, retrasando la puesta en producción del software.
- Facilitar el aprendizaje continuo de todo el equipo, al trabajar de manera integrada.
- Permitir que sea el mismo equipo el que dirige eligiendo (pull) las órdenes que serán realizadas en el flujo de trabajo.

Axioma perseguido

Dado que queremos entregar valor rápidamente, pretendemos limitar la cantidad de trabajo a realizar en cada momento. Queremos finalizar los ítems antes de empezar otros nuevos. Se busca eliminar el tiempo que pasan los ítems esperando innecesariamente.

Ejemplos:

- Se solicita cierta funcionalidad, se incorpora a una lista de solicitudes a ser desarrolladas, ¿cuánto tiempo pasa esa solicitud en la lista sin ser atendida?
- Se analiza una solicitud, ¿cuánto tiempo pasa hasta que la misma es estimada y es asignada para su desarrollo?
- Una vez desarrollado, se encuentra listo para su testeo, ¿cuánto tiempo pasa hasta que la misma es probada?
- Se encuentran errores, ¿cuánto tiempo pasa hasta que los mismos son corregidos?

- El producto está listo, ¿cuánto tiempo pasa hasta que es puesto en producción?

Si eliminàramos esos tiempos, lograríamos que el producto solicitado esté en producción, agregando valor al cliente, mucho más rápidamente. De esa forma, podríamos tomar más rápido otro ítem para iniciar su procesamiento. Entonces lograríamos atender y entregar muchas más solicitudes en la misma cantidad de tiempo.

¿Por qué Kanban?

Se siguen los lineamientos de Kanban para:

- No construir funcionalidades que nadie necesita en este momento.
- No escribir más especificaciones que las que se pueden codificar.
- No escribir más código que el que se puede testear.
- No testear más código que el que se va a entregar.

Costo producido por los atrasos (Pérdidas)

Como vimos, uno de los pilares de Kanban trata de reducir las pérdidas. Ahora bien, ¿cómo interpretamos las pérdidas en un proceso de desarrollo de software? Los aspectos que producen atrasos en el desarrollo son:

- Retrabajo: defectos, fallas , etc.
- Costos operacionales: planificación, generación de release, entrenamiento , etc.
- Costo de coordinación: cronogramas, recursos , etc.

Mediante Kanban, se pretenden minimizar esos costos, a través de trabajar sobre lotes pequeños, solicitudes puntuales, las cuales son rápidamente desarrolladas y que soportan en todo su ciclo de trabajo potenciales cambios, cuya introducción al producto, minimizan los costos relacionados. El mismo proceso está pensado para que la identificación e introducción de cambios sea transparente, con bajo impacto sobre el producto. Solamente se pone el foco en el control de dos parámetros, el tiempo de ciclo y el máximo de solicitudes que pueden ser procesadas por cada estado del ciclo. El proceso a ser seguido es muy sencillo, no demanda importantes entrenamientos y la generación de valor, o releases liberados, se dan de manera continua, con un mínimo impacto en el entorno de trabajo del solicitante.

Dinámica de trabajo

La dinámica de trabajo se da de manera continua, tal cual el flujo de producción lo propone. Las solicitudes van llegando a la lista, en donde se priorizan. El equipo de trabajo va “sacando” de la lista las solicitudes a ser trabajados. Las mismas van fluyendo a través del flujo de trabajo definido. Se controla que no se superen los límites definidos. Si en algún caso (estado) se llega a un límite, el equipo se preocupa por descomprimir los cuellos de botella, por sobre el tomar nuevas solicitudes.
¿Cómo se coordinan las actividades?:
Primero a través de un tablero visible para todo el equipo. Luego a partir de reuniones rápidas de seguimiento, en la cual un facilitador controla los siguientes aspectos:

1. ¿El tablero representa la realidad?
2. ¿Existe alguna solicitud que esté bloqueando el trabajo?
3. ¿Se presentan cuellos de botella en algún estado?
4. ¿Se está cerca de superar algún límite definido?

Luego de la reunión se actualizan los indicadores (tiempo de respuesta y ciclo) y el tablero.

Los Indicadores de Kanban

Los dos indicadores por excelencia de Kanban son tiempo medio de respuesta (lead time) y tiempo del ciclo. El tiempo de respuesta comienza cuando el cliente realiza una solicitud y finaliza cuando la misma le es entregada y está lista para su uso. Es medido en tiempo, no en esfuerzo. Por su parte, el tiempo de ciclo comienza cuando el trabajo sobre la solicitud comienza y finaliza cuando la misma está lista para su entrega.

Figura 3.2 - Indicadores de tiempo en Kanban

Otro aspecto conocido para controlar la performance del proceso es el llamado rendimiento (throughput). Este es el ratio de entregas de valor que se realizan al

cliente durante la producción. Este indicador combina el tiempo de respuesta y ciclo, indicando la cantidad de funcionalidades que pueden ser realizadas en un tiempo dado.
¿Cómo logramos reducir el tiempo de ciclo? A través de las siguientes acciones:

- Reduciendo la cantidad de solicitudes que se encuentran “en proceso”.
- Mejorando la productividad o tiempo de resolución en los diferentes estados (tiempo de procesamiento).
- Reduciendo el re-trabajo.
- Logrando visibilidad de todo aspecto que esté bloqueando el trabajo.
- Logrando que cada solicitud posea un tamaño adecuado, para ser manejada de manera ágil y le agregue valor al cliente. Esto es conocido como MMF: Minimal Markable Feature. Estos MMF son descompuestos en historias, solicitudes cuya información es estructurada en tarjetas (Kanban) y Tareas, para ser administrados a través de un tablero como vimos anteriormente.

Conclusiones

Kanban es otro de los marcos de trabajo que siguen los principios descritos en el manifiesto ágil. A diferencia de otros enfoques, Kanban, se alinea en mayor medida a la idea de flujo continuo de trabajo, donde cada solicitud, luego de ser priorizada, es desarrollada individualmente. A diferencia de ello, por ejemplo SCRUM, incorpora el concepto de lotes que agrupan funcionalidad y definen períodos fijos de tiempo en los cuales los mismos son entregados. Kanban es un excelente marco de trabajo para ser incorporado en todos aquellos casos en los que se presenten algunas de las siguientes características:

- Se requiere rápida respuesta ante cada pedido.
- Se requiere un procesamiento continuo, no ante períodos definidos de tiempo.
- Se posee un equipo de trabajo con capacidad para autogestionar su trabajo y tomar decisiones de compromiso para acelerar los tiempos y las entregas.
- Se deben incorporar características y cambios, en etapas tardías del ciclo de desarrollo.

Capítulo 4

Diseño centrado en el usuario (UCD)

Introducción

El desarrollo centrado en el usuario (UCD) no es realmente un conjunto de metodologías, sino una filosofía o paradigma que un equipo de desarrollo puede seguir. En el mundo de hoy, donde la interacción con la tecnología se convierte en parte de cada vez más tareas diarias, el factor más importante que hará que un servicio sea exitoso, es la satisfacción del usuario final. Esto es principalmente cierto para los productos que las personas quieren usar y no para los productos que las personas tienen que usar. Un ejemplo de esto último son los sistemas empresariales a gran escala que están diseñados para cumplir una función específica, basados en tareas ya conocidas.

"Diseño centrado en el usuario" (UCD) es un término amplio para describir los procesos de diseño en los que los usuarios finales influyen en la forma en que un diseño toma forma. Es a la vez una amplia filosofía y variedad de métodos. Hay un espectro de formas en que los usuarios están involucrados en UCD, pero el concepto importante es que los usuarios están involucrados de una manera u otra. Por ejemplo, algunos tipos de UCD consultan a los usuarios sobre sus necesidades y los involucran en momentos específicos durante el proceso de diseño; Normalmente durante la recolección de requerimientos y pruebas de usabilidad. En el extremo opuesto del espectro, existen métodos UCD en los que los usuarios tienen un profundo impacto en el diseño al participar como socios con diseñadores durante todo el proceso de diseño.
El término 'diseño centrado en el usuario' se originó en el laboratorio de investigación de Donald Norman en la Universidad de California en San Diego (UCSD) en la década de 1980 y se utilizó ampliamente después de la publicación de un libro de co-autor titulado: Diseño del sistema centrado en el usuario: nuevas perspectivas. sobre la interacción humano-computadora (Norman y Draper, 1986). Norman se basó más en el concepto UCD en su libro seminal The Psychology Of Everyday Things (POET) (Norman, 1988). En POET, reconoce las necesidades y los intereses del usuario y se centra en la usabilidad del diseño. Ofrece cuatro sugerencias básicas sobre cómo debe ser un diseño:

- Facilitar la determinación de qué acciones son posibles en cualquier momento.
- Hacer visibles las cosas, incluido el modelo conceptual del sistema, las acciones alternativas y los resultados de las acciones.
- Facilita la evaluación del estado actual del sistema.

- Seguir las asignaciones naturales entre las intenciones y las acciones requeridas; entre las acciones y el efecto resultante; y entre la información que es visible y la interpretación del estado del sistema. (Norman, 1988, p.188)

Estas recomendaciones colocan al usuario en el centro del diseño. El rol del diseñador es facilitar la tarea para el usuario y asegurarse de que el usuario pueda utilizar el producto según lo previsto y con un mínimo esfuerzo para aprender a usarlo. Norman señaló que los manuales largos e incomprensibles que acompañan a los productos no están centrados en el usuario. Sugiere que los productos deben ir acompañados de un pequeño folleto que se pueda leer muy rápidamente y que se base en el conocimiento del mundo del usuario.

El proceso de trabajar con diseño centrado en el usuario (UCD)

La base de UCD es bastante autoexplicativa por su nombre. El objetivo principal de UCD no es siempre tomar decisiones basadas en los requisitos del usuario, sino también involucrar al usuario en el ciclo de desarrollo desde el punto de partida. UCD incorpora los principios de Lean startup con las metodologías de desarrollo ágiles. Lean startup es un método propuesto por Eric Ries y se basa en la fabricación Lean. Ries tradujo esa idea a las startups de software. Para eliminar cualquier práctica derrochadora, las startups magras emplean las siguientes técnicas:
De acuerdo con la metodología Lean, una startup debe crear primero un Producto Mínimo Viable (MVP). Un MVP es la primera versión de un nuevo producto que implementa la funcionalidad más básica que se puede mostrar al cliente lo más rápido posible y recibir comentarios. Esto ayuda a aclarar las suposiciones e hipótesis tempranas para que el equipo tenga una base más firme sobre la cual construir. Una vez que está listo, el equipo recibe comentarios constantes de los usuarios para iniciar la implementación continua en el producto. La implementación continua también asegura que los comentarios de los usuarios se darán pronto para las nuevas características. Una vez que el producto crece en complejidad, las pruebas A / B se utilizan contra grupos de usuarios, para determinar qué características aportan más valor al usuario. Este proceso continuo también se conoce como construir - medir - aprender.
Una parte vital de este proceso es que el desarrollador no se basa en la información obtenida a través de cuestionarios o recopilada por un equipo diferente. Los desarrolladores deben mostrar el producto en persona y luego recopilar información mientras el usuario lo está probando. Esto es especialmente cierto en las primeras etapas de desarrollo, durante las cuales las herramientas como Google Analytics no proporciona datos significativos de un puñado de usuarios. UCD generalmente emplea prácticas ágiles pero, al igual que las características, las prácticas que se consideran desperdiciadas son simplemente eliminadas o reemplazadas por alternativas más fáciles.

Por qué usar diseño centrado en el usuario (UCD)

Es fácil encontrar motivaciones para usar UCD en una startup. Las empresas nuevas generalmente carecen de extensos fondos de dinero, una dirección específica y una base de usuarios. El proceso de UCD ayuda a superar la primera fase de elegir la dirección, a crear software útil con fondos mínimos y a satisfacer la base de usuarios para ayudarla a crecer.
Usaremos un ejemplo de UCD implementado en un software de consumo para mostrar por qué es interesante y necesario su uso.
Supongamos que existe una necesidad de crear un software que provea un servicio para la creación de tickets de bugs online. Supongamos que como primer decisión sobre una característica del software basada en nuestra intuición, asumimos que por ejemplo los tickets van a tener los siguientes estados:

- En definición,
- Pendiente,
- En proceso,
- Hecho,
- En testing,
- Deployado.

Luego de un tiempo y de análisis estadísticos de uso del usuario, el equipo se da cuenta de que no todos los estados de los tickets se utilizan. O que las tareas permanecen sin finalizar de por vida en la base de datos. Entonces se toma la decisión de requerir a un grupo de usuarios que al utilizar el software, indiquen cómo lo usan y qué estados son los que más usan para sus tareas. Esto sería el proceso de interactuar estrechamente con los usuarios del software para obtener comentarios sobre las mejoras necesarias. De ese experimento se obtiene que de todos los estados solo Pendiente, En proceso y Hecho son los más utilizados. El resto no se utilizan o se utilizan muy poco. Este sería un claro ejemplo de características de desperdicio que pueden eliminarse.
Por ello es necesario que, la primera versión del software sirva para las operaciones más básicas que se necesiten, y solo se distribuya a unos pocos usuarios para recibir comentarios. En el transcurso de los años, el proyecto será utilizado por más y más usuarios y se cambiará radicalmente de acuerdo con las ideas de los grupos de usuarios más importantes y los comentarios de boca en boca. Una vez más, esto muestra los conceptos básicos de UCD en acción.
Los beneficios de utilizar UCD son varios.

Los recursos desperdiciados son malos para todos. Podría ser más esencial para las nuevas empresas hacer un uso eficiente de su presupuesto, pero solo porque un equipo tenga más fondos disponibles, no significa que se deban desperdiciar más fondos. Los recursos también significan horas hombre dedicadas a un software.

Según algunas métricas[20], el 45% de las funciones de un software puede que nunca se usen y el 35% se usa rara vez. Esto significa que solo el 20% de las funciones son siempre / frecuentemente utilizadas por los usuarios. Esto es una enorme pérdida de tiempo por parte de los desarrolladores, que podría gastarse mejor en la mejora de las funciones esenciales en lugar de mantener las inútiles.

Los usuarios todavía importan. El éxito del sistema puede no estar basado en el número de sus usuarios, ya que ese número está predeterminado. Pero, ¿por qué querría una empresa que sus usuarios usarán un sistema engorroso lleno de desorden, en lugar de algo que podría tener más sentido para ellos, y hace que el proceso sea más placentero, incluso si faltan algunas características menores (al principio; puede ser implementado más adelante si se considera importante).

Cara a cara con los usuarios. Cuando se crea un sistema empresarial, o cualquier otro sistema, en un grupo grande, los desarrolladores a menudo no tienen tiempo cara a cara con los usuarios finales. En proyectos como estos, es fácil perderse en los asuntos técnicos que pasan los gerentes y olvidar completamente la conversación sobre el usuario final. Un usuario no es solo un conjunto de requisitos y características, sino una persona que incluso podría tener una idea que el desarrollador nunca podría haber pensado. Pasar tiempo con el usuario final es una gran práctica para comprender su punto de vista en cualquier producto.

Aprendiendo nuevas tecnologías. Esto podría ser más un beneficio que una ventaja general de UCD. Pero si un desarrollador tiene que resolver un problema sin herramientas predeterminadas, las posibilidades son infinitas. Tal vez el equipo finalmente se deshaga de la solución costosa (a menudo considerada obsoleta) de usar el servidor SQL solo porque la compañía está asociada con Microsoft. El desarrollador podría descubrir que una base de datos NoSQL puede producir resultados de rendimiento mucho mejores para la tarea requerida y adquirir nuevas habilidades en la implementación de la base de datos.

Introducción a experiencia del usuario Lean (Lean UX)

Existen muchos sucesores de UCD en la actualidad. Los más conocidos y utilizados son Design thinking (2009), Lean UX (2013) y UX Burner (2018).
En este trabajo nos enfocaremos en Lean UX.
Cuando los diseñadores se acercaron al software en los años 80 y 90, lo hicieron de la misma manera en que abordaron los “materiales” anteriores con los que trabajaron. En el diseño industrial, diseño de impresión, diseño de moda y cualquier campo que involucre resultados físicos, el paso de fabricación es una restricción crítica. Al diseñar

[20] http://www.slideshare.net/timmcowan/introduction-to-scrum-3285439

para materiales físicos, los diseñadores deben averiguar qué están haciendo antes de comenzar la producción, porque la producción es costosa.
Trabajando en software, los diseñadores enfrentan nuevos desafíos. Tuvieron que descifrar la gramática de este nuevo medio y, al hacerlo, vieron surgir nuevas especialidades, como el diseño de interacción y la arquitectura de la información. Pero el proceso que los diseñadores practicaban se mantuvo prácticamente sin cambios. Siguieron diseñando productos con gran detalle de antemano, porque aún tenían que lidiar con un proceso de "fabricación": el trabajo tenía que ser duplicado en disquetes y CD, que luego se distribuían al mercado exactamente de la misma manera que los bienes físicos. Repartido. El costo de equivocarse se mantuvo alto.
Hoy se enfrentan a una nueva realidad. Internet ha cambiado la distribución de software de manera radical. La mayoría del software ahora se distribuye en línea. Ya no están limitados por un proceso de fabricación física y están libres para trabajar en ciclos de lanzamiento mucho más cortos.
Pero "gratis" realmente subestima esta nueva realidad. Los equipos ahora enfrentan una intensa presión de los competidores que utilizan técnicas como el desarrollo ágil de software, la integración continua y el despliegue continuo para reducir radicalmente sus tiempos de ciclo. Los equipos están empujando el nuevo código a producción tan rápido como puede guardar un archivo de Photoshop. Y están utilizando estos ciclos cortos como una ventaja competitiva, ya que liberar una versión temprano y con frecuencia, obtienen comentarios del mercado e iteran en función de lo que aprenden, y (quizás de manera inadvertida) aumentan las expectativas de los clientes en términos de calidad y tiempos de respuesta.
En esta nueva realidad, los enfoques tradicionales de "entender todo primero" no son factibles. Entonces, ¿qué deben hacer los diseñadores?
Es tiempo de un cambio. Lean UX (UX = experiencia del usuario) es la evolución del diseño del producto. Toma las mejores partes del kit de herramientas del diseñador y las combina de una manera que las hace relevantes para esta nueva realidad.
Lean UX es profundamente colaborativo y multifuncional, porque los diseñadores no pueden darse el lujo de trabajar aislados del resto del equipo de producto. No pueden seguir pidiéndole a los equipos que esperen a que lo resuelvan todo. Necesitan un compromiso diario y continuo con los equipos para ser efectivos. Este compromiso continuo permite eliminar los productos pesados a favor de las técnicas que permiten construir un entendimiento compartido con los compañeros de equipo.
Lean UX también permite cambiar la forma en que hablamos de diseño. En lugar de hablar sobre características y documentos, podemos hablar sobre lo que funciona. En esta nueva realidad, tienen más acceso a los comentarios del mercado que nunca antes. Esta información permite replantear las conversaciones de diseño en términos de objetivos de negocio. Se puede medir lo que funciona, aprender y ajustar. Lean UX es un enfoque para diseñar y crear experiencias de usuario. En su esencia, trata de validar las hipótesis que se tiene sobre los usuarios y el producto.
Una hipótesis podría leerse así:

"Creo que la gente como persona A tiene una necesidad de (o un problema para hacer) XYZ".

¿Por qué es importante validar las hipótesis en Lean?
Las hipótesis y la validación ayudan a saber si estamos creando productos que la gente quiere. Eso es lo que hace al método Lean. Tenemos espacio para pivotar y cambiar.
Se podría pensar que no hablar con sus usuarios (o usuarios potenciales) es algo desastroso, pero muchas compañías todavía están creando productos que nadie quiere y perdiendo una tonelada de dinero en el proceso. De hecho, el 42% de las nuevas empresas fracasan porque no hay necesidad del mercado.
Lean UX es una metodología que ayuda a evitar la fabricación de productos que nadie quiere. Laura Klein escribe en su libro Lean UX for Startups:

“En lugar de pensar en un producto como una serie de características que se construirán, Lean UX considera un producto como un conjunto de hipótesis para validar. En otras palabras, no asumimos que sabemos lo que quiere el usuario ".[21]

Hay mucha superposición con el Diseño Centrado en el Usuario (UCD) y Lean UX. UCD es iterativo como Lean y ambos procesos se enfocan en el usuario y sus necesidades en cada fase del proceso de diseño. UCD también involucra al usuario en el proceso de diseño, al igual que Lean UX.

[21] UX for Lean Startups By Laura Klein

El proceso de Lean UX

El proceso de Lean UX, puede puede verse como un refinamiento de las etapas comunes que se aplican en Design Thinking[22]:

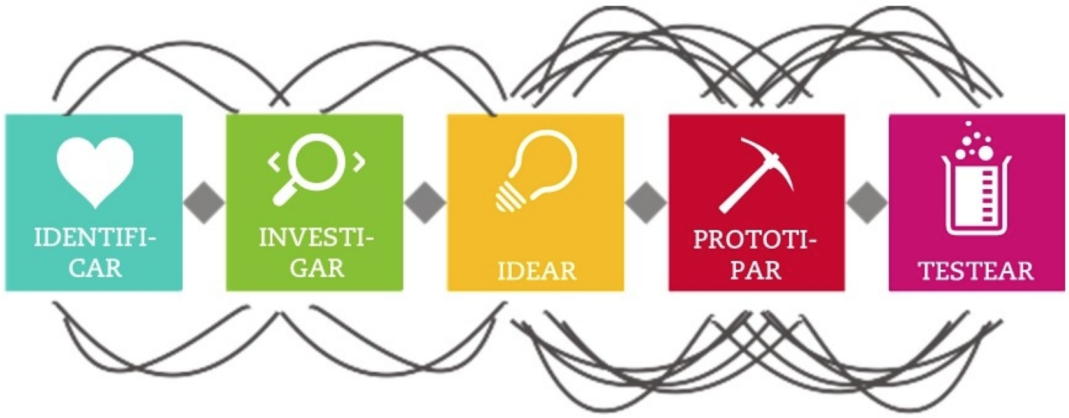


Figura 4.1 - Etapas de la Metodología Design Thinking

En concreto la Metodología Design Thinking propone 5 pasos principales

- Identificar: Definir el problema que se va a tratar
- Investigar: Recoger información relacionada con el problema a tratar
- Idear: generar ideas en torno a la información que interpretamos
- Prototipar: Materializar las ideas para llevarlas al mundo físico
- Testear: Probar los prototipos con los usuarios para aprender y refinar la soluciones

Lean UX se basa en 3 conceptos principales: pensar, hacer y verificar (Think, Make Check en Inglés).

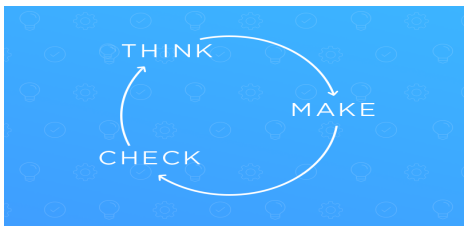


Figura 4.2 - Bucle de iteración UX

[22] **Pensamiento de diseño (Design thinking)** hace referencia a los procesos cognitivos, estratégicos y prácticos mediante los cuales se elaboran los conceptos relacionados con el diseño (propuestas de nuevos productos, edificios, máquinas, etc.). Muchos de los conceptos y aspectos fundamentales del pensamiento de diseño han sido definidos por medio de estudios, en diferentes áreas del diseño, de la cognición del diseño y de la actividad de diseño tanto en laboratorios como en entornos naturales.

Pensar se refiere a crear suposiciones acerca de cualquier problema dado y lo que creés saber sobre esa área en particular.

Estas suposiciones son importantes porque, eventualmente, brindan nuevos conocimientos que se convertirán en hipótesis para ser probadas. Otros aspectos de la etapa de pensar implican:

- Investigación
- Ideación
- Modelos mentales
- Bocetos
- Storyboards

Hacer implica diseñar un Producto Mínimo Viable (MVP). Un MVP es lo mínimo que necesita para atraer a su audiencia e iniciar un ciclo de retroalimentación de iteración. Un MVP podría ser una página de destino (landing page) de un producto que aún no existe para medir el interés del cliente.
Esta etapa es iterativa (es decir, se hace lo mínimo y se continúa construyendo sobre ella). Aquí hay otras formas en que puede crear su propio MVP:

- Prototipos
- Wireframes
- Propuestas de valor
- Páginas de destino
- Hipótesis
- Código desplegado

“Lean UX es donde brilla la creación de prototipos. Al igual que con los bocetos iniciales, enfocar el prototipo en los componentes críticos de la experiencia es esencial ". - Jeff Gothelf, Smashing Magazine[23].

Check es la última etapa en el bucle Lean UX. Aquí es donde se prueba el MVP con la audiencia y lo valida o lo invalida, llevando el nuevo conocimiento a la etapa de reflexión para comenzar de nuevo. Para la etapa de Check se puede utilizar:

- Pruebas A / B
- Analítica del sitio
- Pruebas de usabilidad
- Sistema de retroalimentación
- Reuniones de clientes

[23] Jeff Gothelf - https://www.smashingmagazine.com/author/jeff-gothelf/

Los principios fundamentales de la metodología Lean UX

Suposiciones

En Lean UX, los supuestos desempeñan un papel fundamental, merecen ser comprendidos un poco más.
La razón por la que los supuestos desempeñan un papel importante en Lean UX es que hay un cambio respecto al método tradicional de requisitos / entregables. En lugar de confiar en los requisitos, el enfoque se centra en crear una declaración de problema.
Una declaración de problema puede tomar esta forma:

*Hemos observado que **[producto / servicio / organización]** no cumple con **[estos objetivos / necesidades]**, lo que está causando **[este efecto adverso]**. ¿Cómo podríamos mejorar para que nuestro producto / servicio / equipo / organización tenga más éxito en base a **[estos criterios mensurables]**?*

Esto te ayuda a saber cuándo has resuelto el problema. Es probable que cuando se comience a trabajar de esta manera, tengamos varias declaraciones de problemas. Es responsabilidad del equipo decidir qué declaraciones de problemas son más pertinentes para los objetivos comerciales y de los usuarios y seguir esa ruta.
A partir de la declaración del problema, se puede comenzar a hacer suposiciones. Al crear suposiciones, es mejor tener la declaración del problema en frente. Porque luego es necesario pensar en necesidades, usuarios y soluciones.

Existen algunas frases como estas para guiarse mientras se crean suposiciones:

- Mi cliente necesita ...
- Creo que adquiero la mayoría de mis usuarios a través de ...
- El mayor riesgo es ...
- ¿Qué características son importantes?
- ¿Qué problema resuelve el producto?

Hipótesis

Después de escribir una declaración de problema, identificar y seleccionar las suposiciones principales, la siguiente etapa es desarrollar una hipótesis.
Para probar un supuesto, simplemente se convierten en una hipótesis. Una plantilla de declaración de hipótesis se ve así:

*Creemos que este **[resultado empresarial]** se logrará si **[estos usuarios]** alcanzan con éxito **[logran este resultado de usuario]** con esta **[característica]**.*

Y esto es lo que podría parecer cuando se haya creado una hipótesis real:

Creemos que las conversiones aumentarán en un 25% (resultado del negocio) si los nuevos clientes (su público objetivo) identifican con éxito un botón de llamada a la acción más grande (función).

Una vez que ha creado una declaración de hipótesis, es hora de ponerla a prueba. Como se puede ver en la declaración de hipótesis, hay que comenzar con lo que se espera que suceda (más conversiones), se continúa con quién ayudará a lograrlo (nuevos clientes) y cómo (botón grande y brillante de llamada a la acción).
Lo más interesante de crear una hipótesis es que si alguna vez se escucha a alguien decir el temido "No creo que esta idea funcione", se puede poner a prueba y descartarla o perseguirla, sofocando efectivamente cualquier lucha interna o desacuerdos.

Producto mínimo viable

El Producto mínimo viable (MVP) es un concepto central en Lean UX. La idea es construir la versión más básica del concepto como sea posible, probarlo y si no hay resultados valiosos, abandonarlo. Los MVP que se muestran prometedores pueden incorporarse a otras rondas de diseño y desarrollo sin demasiada molestia.
Al crear un MVP, se está creando un pequeño producto que funciona. No se necesitan luces destellantes para un MVP. El MVP puede ser tan sencillo que solo una página de destino puede ser suficiente. Solo se necesita tener algo para satisfacer a los primeros clientes y ganar interés.

"Al considerar la creación de su propio producto mínimo viable, deje que esta simple regla sea suficiente: elimine cualquier característica, proceso o esfuerzo que no contribuya directamente al aprendizaje que busca". Eric Ries, The Lean Startup

Testing

La prueba es la columna vertebral de Lean UX. Realizar pruebas es lo que da respuestas. Ayuda a comprender por qué los usuarios interactúan con el producto de la forma en que lo hacen.
Guiará las decisiones de diseño y ofrecerá claridad. Las pruebas tampoco tienen que involucrar a mucha gente. Se puede hacer con 3-5 personas así como con personal interno.
La prueba no moderada es Lean porque es barata de hacer, rápida para encontrar reclutas y se puede escalar. Puede utilizar una variedad de artefactos de prototipos de baja fidelidad, HTML y wireframes seleccionables. Cuanto menor sea la fidelidad, mejor porque se tiene un producto para lanzar y no hay tiempo que perder.
Las pruebas no moderadas solo significan que un usuario puede completar una sesión de pruebas sin necesidad de un investigador de UX en la sala. Esto se puede

hacer en un servicio como Userzoom[24] y los usuarios pueden hacerlo desde la comodidad de su hogar.

Conclusión

Diseño centrado en el usuario (UCD) es un término general para una filosofía y métodos que se centran en el diseño para los usuarios y la participación en el diseño de sistemas informáticos. Las formas en que los usuarios participan pueden variar. En un extremo del espectro la participación puede ser relativamente ligera; Se les puede consultar sobre sus necesidades, observarlos y participar en las pruebas de usabilidad. En el otro extremo del espectro, la participación puede ser intensiva con los usuarios que participan en todo el proceso de diseño como socios en el diseño. Se han desarrollado una variedad de métodos para apoyar la UCD, incluidas las pruebas de usabilidad, ingeniería de usabilidad, evaluación heurística y diseño participativo. Evaluaciones más ligeras también son importantes, ya que las ideas se llevan a unos pocos usuarios representativos por sus comentarios al inicio del diseño. Se ha demostrado que involucrar a los usuarios en el diseño de una forma u otra conduce al desarrollo de diseños satisfactorios más utilizables.
Usar UCD es esencial para el conocimiento del usuario. Las startups luchan por realizar el ajuste producto-mercado correcto. Su producto es a menudo algo que los usuarios ni siquiera saben que necesitan. Steve Jobs decia que: “It’s not the customer’s job to know what they want”. Traducido quiere decir que “NO es tarea de los usuarios saber qué quieren”. Ellos saben cuales son sus problemas o necesidades, cuáles son sus tareas recurrentes, cuáles son los datos frecuentemente utilizados pero nada saben sobre cuál es la mejor solución a esos problemas. Eso es tarea de los expertos y profesionales tecnológicos y en el caso de diseño de interfaces, también diseñadores gráficos y multimediales. Por eso es necesario utilizar una metodología que permita conectar lo que el usuario quiere con la solución.
Cuando, por ejemplo, recibimos comentarios sobre uno de nuestros productos, tenemos que esperar que estas sean solo sugerencias para mejorar, en lugar de un problema importante que hayan encontrado. Tenemos que asumir absolutamente que antes de dar vida a nuestro producto, la investigación de usuarios, las pruebas de usuarios y las revisiones de expertos han eliminado todos los problemas de UX. Cualquier sugerencia de mejoras generada por el cliente debe ser estudiada cuidadosamente por nuestros expertos y expuesta a la importante pregunta "¿Por qué necesitamos esta característica en nuestro producto?".
UCD es muy amplio y se han creado muchas variantes bastante modernas, como son Design Thinking, Lean UX y el más innovador y moderno UX Burner. Lean UX es un cambio de paradigma. Crea eficiencia y elimina cualquier desperdicio, a diferencia de otras metodologías de diseño comunes.

[24] Userzoom - https://www.userzoom.com/es/

Al reducir la dependencia de los entregables y al concentrarse en pequeñas ganancias a lo largo del tiempo, los productos pueden llegar al mercado más rápido sabiendo que su cliente realmente quiere lo que tiene que vender.
Cualquier equipo de productos orientado al diseño debe usar Lean si desean mejorar su flujo de trabajo y mejorar la experiencia del usuario de sus productos.
En este trabajo nos interesa aplicar los conceptos de Lean UX.

Capítulo 5

ReactJs

Qué és ReactJs y para que fue concebido

React es una librería Javascript focalizada en el desarrollo de interfaces de usuario. Esa es su principal área de trabajo, pero lo cierto es que con todo el ecosistema de aplicaciones y herramientas y componentes, con React encontramos un excelente aliado para hacer todo tipo de aplicaciones web, SPA (Single Page Application) o incluso aplicaciones para móviles. Es por tanto una base sobre la cual se puede construir casi cualquier cosa con Javascript y que nos facilita mucho el desarrollo, ya que nos ofrece muchas cosas construidas, en las que no necesitamos invertir tiempo para desarrollar. React es una librería que proviene de Facebook. Es software libre y a partir de su liberación una creciente comunidad de desarrolladores la está usando. Se crea en base a unas necesidades, generadas por el propio desarrollo de la web de la popular red social. En Facebook necesitaban herramientas para un desarrollo rápido pero focalizadas en un mayor rendimiento que otras alternativas existentes en el mercado. Detectaron que el típico marco de binding y doble binding ralentizaba un poco su aplicación, debido a la cantidad de conexiones entre las vistas y los datos. Como respuesta crearon una nueva dinámica de funcionamiento, en la que optimizaron la forma de cómo las vistas se renderizan frente al cambio en los datos de la aplicación. A partir de ahí la probaron en su red social con resultados positivos y luego en Instagram, también propiedad de Facebook, con éxito. Más adelante, alentados por los positivos resultados en el rendimiento de React, muchas otras aplicaciones web de primer nivel la fueron adoptando. BBC, Airbnb, Netflix, Dropbox y un largo etc.
Sirve para desarrollar aplicaciones web de una manera más ordenada y con menos código que si usas Javascript puro o librerías como jQuery centradas en la manipulación del DOM. Permite que las vistas se asocian con los datos, de modo que si cambian los datos, también cambian las vistas.
El código “spaghetti” que se suele producir mediante librerías como jQuery se pretende arquitecturizar y el modo de conseguirlo es a través de componentes. Una interfaz de usuario es básicamente creada a partir de un componente, el cual encapsula el funcionamiento y la presentación. Unos componentes se basan además en otros para solucionar necesidades más complejas en aplicaciones. También permite crear otras piezas de aplicación cómodamente, como los test.

Por qué elegir ReactJs

React es isomórfico

Éste es un concepto relativamente nuevo, pero muy interesante en el desarrollo de aplicaciones que se desean tengan un buen posicionamiento en buscadores. Básicamente se trata de que, con un mismo código, se pueda renderizar HTML tanto en el servidor como en el cliente, rebajando la carga de trabajo necesaria para realizar aplicaciones web amigables para buscadores.El problema de las aplicaciones Javascript es que muchas veces reciben los datos en crudo del servidor, o un API, en formato JSON. Las librerías Javascript y frameworks toman esos datos para producir el HTML que debe representar el navegador. Esto, que es la solución más adecuada, porque nos permite desacoplar el servidor del cliente, pero se convierte en un aspecto negativo de cara al posicionamiento en buscadores como Google, debido a que el cuerpo de la página no tiene contenido. Al no tener contenido una página que recibe los datos en un JSON, Google no sabe qué palabras clave son interesantes y no otorga ranking para ellas. Con ello la aplicación o página no consigue posicionar. Google está haciendo cambios y ha comenzado a procesar el Javascript para saber los datos de una página, pero aún dista de las ventajas que supone que el contenido esté en el propio HTML que entrega el servidor. Para la librería que nos ocupa, React, permite isomorfismo. Eso es que, con el mismo código somos capaces de renderizar tanto en el cliente como el servidor. Por tanto, cuando llega un buscador como Google, con la misma base de código se le puede entregar el HTML con el contenido ya renderizado, lo que lleva a que una aplicación React sea capaz de posicionarse tan bien como una aplicación web tradicional que renderice del lado del servidor, como es el caso de un desarrollo tradicional o un desarrollo basado en un CMS como WordPress. [3] [16]

Porque tiene un ecosistema muy Rico

React en sí es una librería y como tal hay cosas que se deja del lado de fuera con respecto a soluciones aportadas por los frameworks MVC. Sin embargo existe todo un ecosistema de herramientas, aplicaciones y librerías que al final equiparan React a un framework. Hay herramientas que se usan en múltiples proyectos, como el caso de Redux[25] o Flux[26], que aportan partes que React no se encarga. Éstos se ocupan del flujo de datos en React y lo resuelven de una manera optimizada, elegante, poniendo énfasis en la claridad de las aplicaciones. Como desarrolladores podemos escoger entre varios frameworks encargados del flujo de los datos, basados en React. Como otros ejemplos tenemos generadores de aplicaciones, sistemas de routing del lado del cliente, etc.
Por otra parte, al desarrollar en base a componentes reutilizables permite que puedas usar el desarrollo de un proyecto en otro. Y por el mismo motivo, encuentras una

[25] Redux - https://es.redux.js.org/
[26] Flux - https://facebook.github.io/flux/

amplia comunidad que libera sus propios componentes para que cualquier persona los pueda usar en cualquier proyecto. Por tanto, antes de desarrollar algo en React conviene ver si otro desarrollador ya ha publicado un componente que lo haga y en la mayoría de los casos, cuando se trata de cosas de ámbito general, veremos que siempre es así.
Hay componentes desde simples botones, sliders, tooltips, etc. Es muy sencillo que se pueda compartir, gracias a que los componentes son capaces de trabajar de manera independiente y que encapsulan funcionalidad para que no interaccionen con otros componentes si no se desea. [16]
React Native[27] es otra de las herramientas disponibles en el ecosistema, que permite llevar una aplicación escrita con Javascript y React como aplicación nativa para dispositivos iOS, Android, etc. Y se trata de aplicaciones nativas, no de web views. [4]

Porque es orientado a componentes

Cuando pensamos en la solución a un problema, sabemos que si lo dividimos en problemas más pequeños y generamos soluciones para cada uno de los mismos, obtendremos la solución al problema general de una manera mas ordenada y rápida. Es más fácil pensar las soluciones si utilizamos la técnica del “divide y vencerás”. Donde a un problema grande lo dividimos en problemas pequeños y buscamos soluciones puntuales para esos problemas. Cuando se diseña una interfaz de usuario, que es la solución general al problema, es más complicado modularizar la solución si no podemos ver cuáles son los componentes lógicos de ese diseño, y cómo interactúan entre sí.
Un componente es una unidad reutilizable, con una interfaz bien definida. Contiene en sí todo lo necesario para su funcionamiento, con ninguna o muy pocas dependencias con otros componentes o librerías. Es tarea del programador en todos los casos, analizar el diseño y ver qué partes del diseño son componentes que puedan reutilizarse, o simplemente ser creados para, trabajar enfocados en la solución a componentes más pequeños. Luego los componentes interactúan entre sí para solucionar el problema principal que resuelve esa interfaz de usuario.

Reutilización de componentes

Bueno, esto ha llegado como una de las mayores bendiciones para los desarrolladores. Realmente se pueden reutilizar los componentes que se desarrollaron en otra aplicación que comparte la misma funcionalidad.
Esto ayuda enormemente a ahorrar esfuerzos y tiempo, y se puede completar el proyecto a un ritmo mucho más rápido.
Además, dado que hay menos codificación involucrada con ReactJS, hay menos posibilidades de crear errores. Se generan menos puntos de fallo al reutilizar componentes que ya fueron desarrollados y probados en otros contextos.

[27] React Native - https://facebook.github.io/react-native/

La complejidad de los componentes que se puede reutilizar es muy variada. Desde simple botón, hasta un componente personalizado con comportamiento más complejo. Por supuesto, también se reutilizan los componentes que desarrolla la comunidad de desarrolladores de ReactJs.

ReactJS es mucho más fácil de aprender

Bueno, aquí también, hay una situación de ganar-ganar para los nuevos desarrolladores. Si eres nuevo en este campo, sería más fácil escalar una montaña, ya que aprender React JS se considera más sencillo en comparación con los demás. Puede acceder fácilmente a la información y familiarizarse con sus características para desarrollar aplicaciones web y móviles. ReactJS no cuenta con todas las funciones, pero tiene la ventaja de que la biblioteca de la GUI de JavaScript de código abierto ayuda a ejecutar la tarea de una mejor manera.
También se puede conocer como la V en el patrón MVC (Controlador de vista de modelo). Por lo tanto, puede dominar el React JS en un corto período de tiempo para crear aplicaciones web altamente sensibles.

Mejora del rendimiento debido a DOM virtual

El Document Object Model o DOM es una plataforma multiplataforma y de programación que se ocupa de HTML, XML o XHTML como una estructura de árbol en la que cada uno de los nodos representa una parte del documento. Pero la mayoría de los desarrolladores enfrentaron el problema cuando se actualizó el DOM, ya que ralentiza el rendimiento.

Sin embargo, ReactJS tiene una solución al introducir el DOM virtual. Puede crear un DOM virtual con ReactJS y alojarlo en la memoria.
El beneficio es que cuando se produce algún cambio en el DOM real, el DOM virtual también tiende a cambiar. Y el DOM tampoco se actualiza con frecuencia, lo que lleva a un rendimiento más suave y más rápido.

Diferencia entre ReactJs y frameworks convencionales

Con respecto a librerías sencillas como jQuery, React aporta una serie de posibilidades muy importante. Al tener las vistas asociadas a los datos, no necesitamos escribir código para manipular la página cuando los datos cambian. Esta parte en librerías sencillas es muy laboriosa de conseguir y es algo que React hace automáticamente.
También en comparación con jQuery nos permite una arquitectura de desarrollo más avanzada, con diversos beneficios como la encapsulación del código en componentes, que nos ofrecen una serie de ventajas más importantes que los plugin, como la posibilidad de que esos componentes conversen e interaccionen entre sí, algo que que sería muy difícil de conseguir con Plugins.
ReactJS solapa por completo las funcionalidades de jQuery, por lo que resulta una

evolución natural para todos los sitios que usan esa librería. Podrían convivir pero no es algo que realmente sea necesario y recargaría un poco la página, por lo que tampoco sería muy recomendable.

Ya luego en comparación con frameworks como es el caso de Angular o Ember, React se queda a mitad de camino. A partir de todo el ecosistema de React se llega más o menos a las mismas funcionalidades, así que es una alternativa perfecta. Decimos que se queda a mitad de camino porque React por sí mismo es una librería y no un framework, puesto que React se ocupa de las interfaces de usuario. Quizás nos sirva decir que sería la "V" en un framework "MVC", aunque es solo una manera de hablar, puesto que React podría ocupar también parcelas de lo que sería la "C". Todo depende de nuestra manera de trabajar aunque, no obstante, esta posible carencia con respecto a los frameworks Javascript se soluciona con capas adicionales a React. Lo que podría interpretarse como una desventaja, muchos desarrolladores lo entienden como una ventaja con respecto a frameworks completos, ya que tú puedes desarrollar con React a tu gusto, aplicando aquellas herramientas y librerías adicionales que hacen las cosas como mejor se adapte al proyecto. No se puede decir de una manera objetiva si es ReactJS es mejor o peor que otras alternativas, porque eso ya entra más en el terreno de la opinión. Lo cierto es que muchas librerías se especializan en el "data-binding", pero React toma esa misma necesidad y la resuelve de otra manera. La diferencia es que React le pone más inteligencia a la necesidad de actualizar una vista cuando es necesario y lo consigue mediante el "DOM Virtual". En líneas generales el virtual DOM es una representación del DOM pero en memoria. Cuando se actualiza una vista se actualiza el DOM Virtual, que es mucho más rápido que actualizar el DOM del navegador. Cuando React compara el DOM Virtual con el DOM del navegador sabe perfectamente qué partes de la página debe actualizar y se ahorra la necesidad de actualizar la vista entera. Es algo muy potente y es la base del rendimiento optimizado de React. Esto se hace de manera transparente para el desarrollador, que no necesita intervenir en nada.Hoy en día, la web es diferente. La forma en que construimos para la web es diferente. Enfrentados a los desafíos de tratar con de mantener el inentendible e impensable código que jQuery produjo, se tuvieron que buscar nuevas formas de administrar la complejidad de las interfaces de usuario modernas. Se necesitaba una nueva biblioteca de interfaz de usuario que ayudará a crear aplicaciones frontend declarativas, modulares, rápidas y escalables utilizando JavaScript. React.js, aporta ideas profundas sobre cómo trabajar con el DOM, organizar el flujo de datos de su aplicación y pensar en los elementos de la interfaz de usuario como componentes individuales. Y, sin embargo, es sólo una biblioteca de interfaz de usuario que no hace suposiciones sobre el resto de la tecnología subyacente. [16]

React y su JSX

JSX es una extensión de JavaScript creada por Facebook para el uso con su librería React. Sirve de preprocesador y transforma el código a JavaScript.
Puede parecer que se está mezclando código HTML con JavaScript, pero nada más lejos de la realidad.
React al basar el desarrollo de apps en componentes, se necesitan crear elementos HTML que definen el componente, por ejemplo , <p>, <img>, etc.
También se necesita indicar cuando se trata de componentes creados con React, como puede ser un <Image />, <List />, <MyComponent> etc.
Todo esto se puede hacer con JavaScript con los métodos que ofrece React como React.createElement en su API. Por ejemplo:
Cear un componente <Icon /> que está definido por un div, un img y algunas clases de CSS. Con JavaScript sería algo así:

```
var image = React.createElement('img', {
  src: 'react-icon.png',
  className: 'icon-image'
});

var container = React.createElement('div', {
  className: 'icon-container'
}, image);

var icon = React.createElement('Icon', {
className: 'avatarContainer'
}, container);

ReactDOM.render(
  icon,
  document.getElementById('app')
);
```

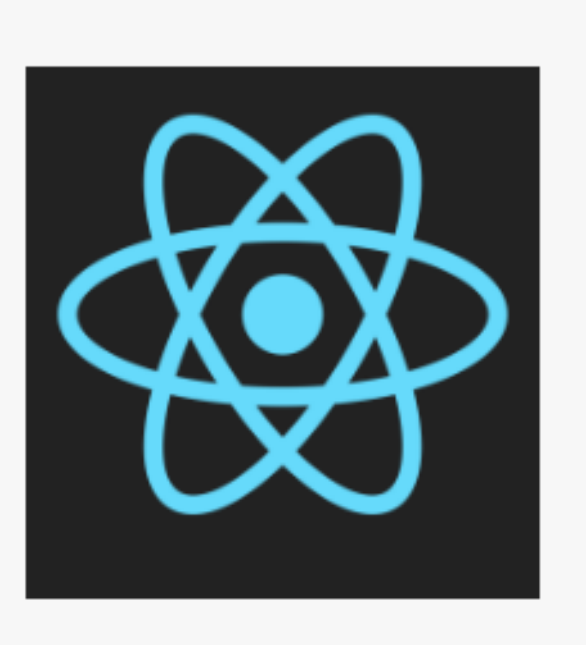
Con esto tendríamos un componente <Icon /> que se traduciría al siguiente código HTML:

```
<div class='icon-container'>
  <img src='icon-react.png' class='icon-image' />
</div>
```

Si tuviésemos el siguiente CSS:

```
.icon-image                                                    {
  width:                                                     100px
}
.icon-container                                                {
  background-color:                                          #222;
  width:                                                     100px
}
```

El resultado en el navegador sería así:



Ahora veamos como se haría lo mismo pero empleando sintaxis JSX:

```
var Icon = (
  <div className='icon-container'>
    <img
      src='icon-react.png'
      className='icon-image'
    />
  </div>)

ReactDOM.render(Icon, document.getElementById('app'))
```

Como se puede ver, es mucho más práctico y legible esta sintaxis. Es prácticamente como escribir HTML pero no estás escribiendo HTML, es JavaScript.

Lo único que se debe de tener en cuenta es que hay algunas palabras reservadas en JavaScript y JSX te obliga a nombrar algunos atributos de otra manera, como es el caso de la palabra **class** que para definir clases de CSS que con JSX debemos escribir **className**.
A medida que nuestra aplicación va creciendo y tenemos componentes más grandes, que manejan distintos eventos, esta forma de usar JSX nos va a ayudar mucho a agilizar nuestros desarrollos.

Capítulo 6

Propuesta de marco de trabajo

Objetivos

Cuando desarrollamos un software en alguna tecnología que sea open source, donde la comunidad provea de funcionalidad al core a través de plugins, estamos utilizando una porción de cada uno de ellos para elaborar nuestra solución. En otras palabras, utilizamos lo esencial, lo que realmente nos sirve para concretar la solución. Podremos tener el framework más completo del mundo listo para desarrollar, pero lo que determina qué usaremos de él va a ser el dominio de la solución y las necesidades que haya que satisfacer con ese software. Por ello, todo el tiempo cuando implementamos soluciones, tomamos y usamos lo esencial.
Análogamente, esta propuesta tiene como filosofía tomar lo esencial de cada uno de los temas que fueron presentados en este documento y elaborar con ellos un marco de trabajo.
A muy alto nivel, Lean UX es un enfoque para explorar problemas complejos y encontrar nuevas oportunidades centrados en las personas. Lean es una filosofía de gestión que usa el razonamiento científico para explorar y validar nuestras asunciones e hipótesis para mejorar el flujo de valor de punta a punta en las organizaciones. Para hacerlo procede de una forma experimental, construyendo prototipos para exponerlos frente a los clientes en ciclos de aprendizaje validado, ajustando las próximas acciones en función de lo aprendido.
Kanban funciona como un organizador de proyectos, tareas y sus estados para trabajar en equipo. Y por último, ReactJs facilita la construcción de las soluciones utilizando un filosofía orientada a componentes reutilizables.
Dado que tomaremos lo esencial de cada tema, los objetivos principales de este marco de trabajo son los siguientes:

- Agilizar la resolución de proyectos, reduciendo los tiempos de análisis, desarrollo y pruebas.
- Mantener al equipo de diseño y desarrollo enfocados y trabajando en conjunto con los clientes o usuarios.
- Empoderar a los clientes o usuarios finales del proyecto, para que ellos siempre sean el foco de las decisiones.
- Utilizar la filosofía Lean como eje para la toma de decisiones.
- Poder dar valor al cliente a través de un flujo de valor lo más rápido posible para ir mejorando los procesos continuamente.
- Tener un mecanismo de comunicación ágil en el equipo de trabajo.
- Estructurar las tareas del proyecto lo más entendible y ágil posible.

Alcance

Para simplificar el alcance de la propuesta, la misma va a estar orientada a trabajar con software de consumo, eso significa que tienen que estar a disposición de clientes o usuarios a través de internet, y que no es un software empresarial o de propietario, que no permita el registro de nuevos usuarios de una manera dinámica, como tampoco la creación de nueva funcionalidad basada en suposiciones o hipótesis.
El software además, tendrá que ser nuevo. Es decir, el marco se enfocará para la aplicación de software creado desde cero, y no para software existente.
Deberá ser orientado a servicios, utilizando a internet como medio de disponibilización.
Por último podemos decir que el marco está pensado para el software que se utilice como SAAS (Software as a Service).
La aplicación del marco de trabajo se hará utilizando un proyecto e idea de ejemplo que será inventado exclusivamente para la realización de este trabajo.

Definición del marco de trabajo

Paso 1: Definir hipótesis

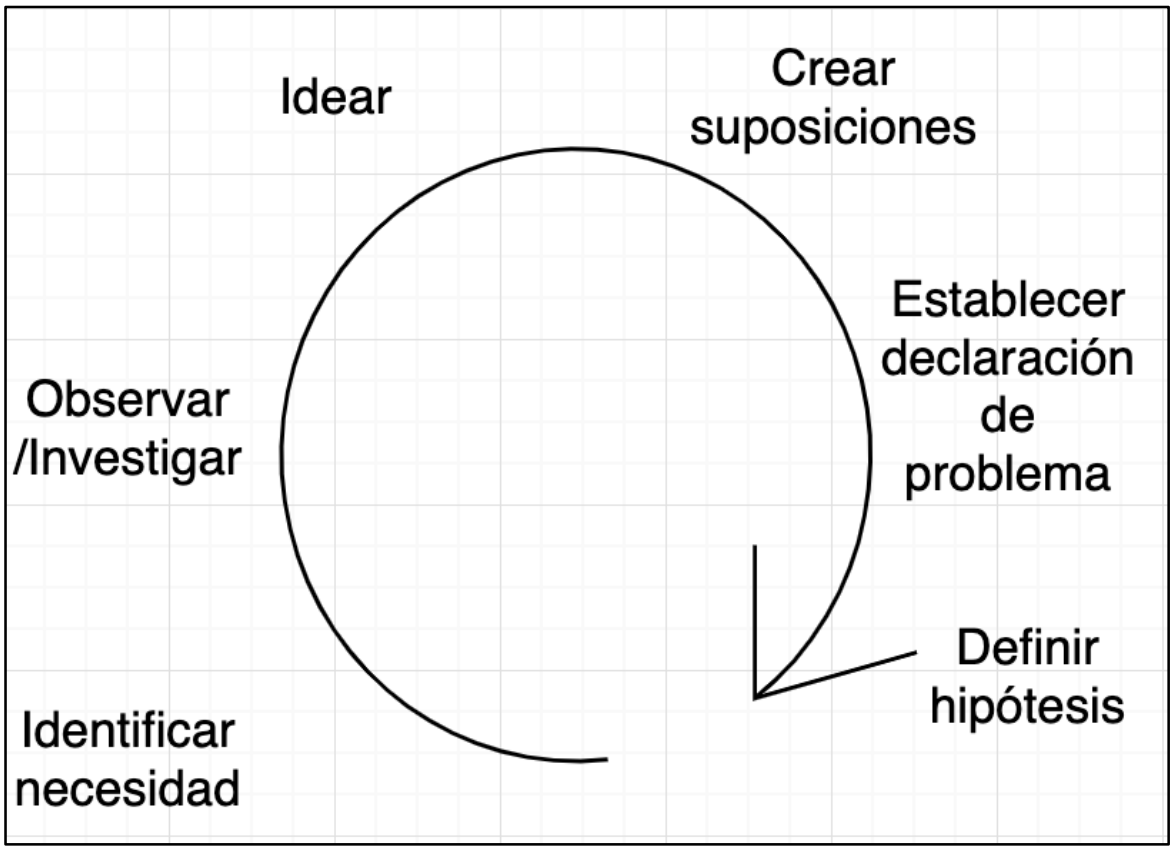


Figura 6.1: Paso 1 del marco de trabajo.

Este paso simplemente consiste en los 3 primeros pasos de la metodología Lean/UX:

- Identificar: Definir el problema que se va a tratar.
- Investigar: Recoger información relacionada con el problema a tratar.
- Idear: Generar ideas en torno a la información que interpretamos.

Se pueden hacer suposiciones o predicciones de comportamiento a futuro de los usuarios para elaborar nuevas hipótesis en forma de afirmación. De esta manera tenemos un punto de partida, al seleccionar una de ellas para validar. La selección de la hipótesis dependerá del contexto del proyecto y de las priorizaciones de las tareas en desarrollo y pendientes. Es importante tener presente que en este tipo de modelo, no existen requisitos / entregables. En lugar de confiar en los requisitos, el enfoque se centra en crear una declaración de problema y luego una hipótesis que considere a la declaración como resuelta.

El resultado de este paso es la hipótesis definida.

Paso 2: Definir MVP

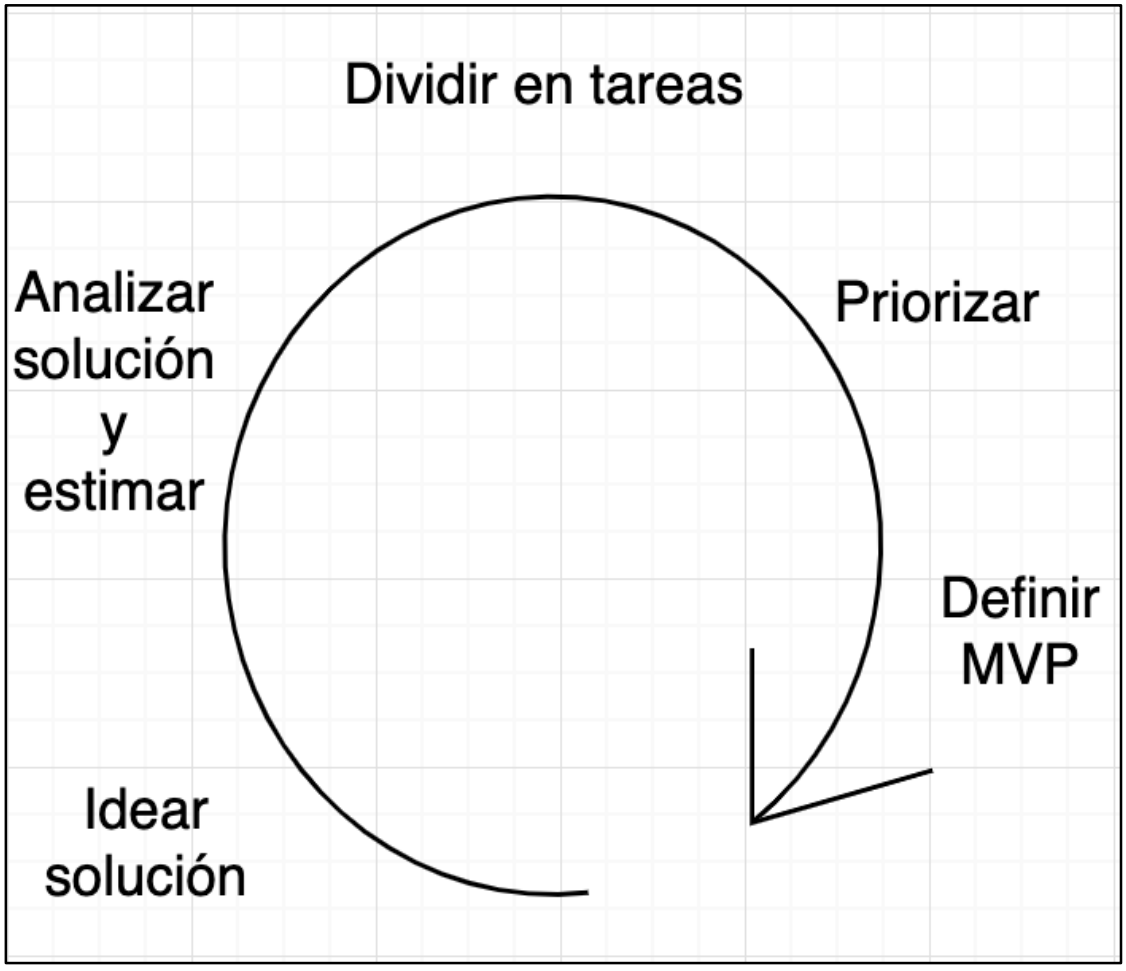


Figura 6.2 : Paso 2 del marco de trabajo

En este punto decidiremos qué usaremos para validar nuestra hipótesis y cómo lo haremos visible a los usuarios. Como se definió en el alcance de este trabajo, este marco este marco está pensado para ser aplicado en software de consumo disponible a los usuarios a través de internet. Por lo tanto nuestro MVP debe cumplir ese requisito. La idea principal es “crear la versión de un nuevo producto que permite a un equipo recoger con el mínimo esfuerzo la máxima cantidad de conocimiento validado acerca de los consumidores” [15]. Entonces es necesario que la primera versión del software sirva para las operaciones más básicas que se necesiten, y solo se distribuya a unos pocos usuarios para recibir los primeros suscriptores o usuarios registrados. Comúnmente denominados Early Adopters [22]. Los primeros adoptantes, que así se denominan en castellano, son aquellos consumidores a quienes la propuesta como marca le aporta una solución a un problema, incluso si la solución no es del todo perfecta. Incluso es posible que puedan ayudar con un aporte de conocimiento y feedback para la mejora de la propuesta.
El primer adoptante es aquel que primero prueba, el atrevido, el que tiene hambre voraz de novedad. Esa voracidad de nuevas experiencias genera que sean hábiles detectores de tendencias.

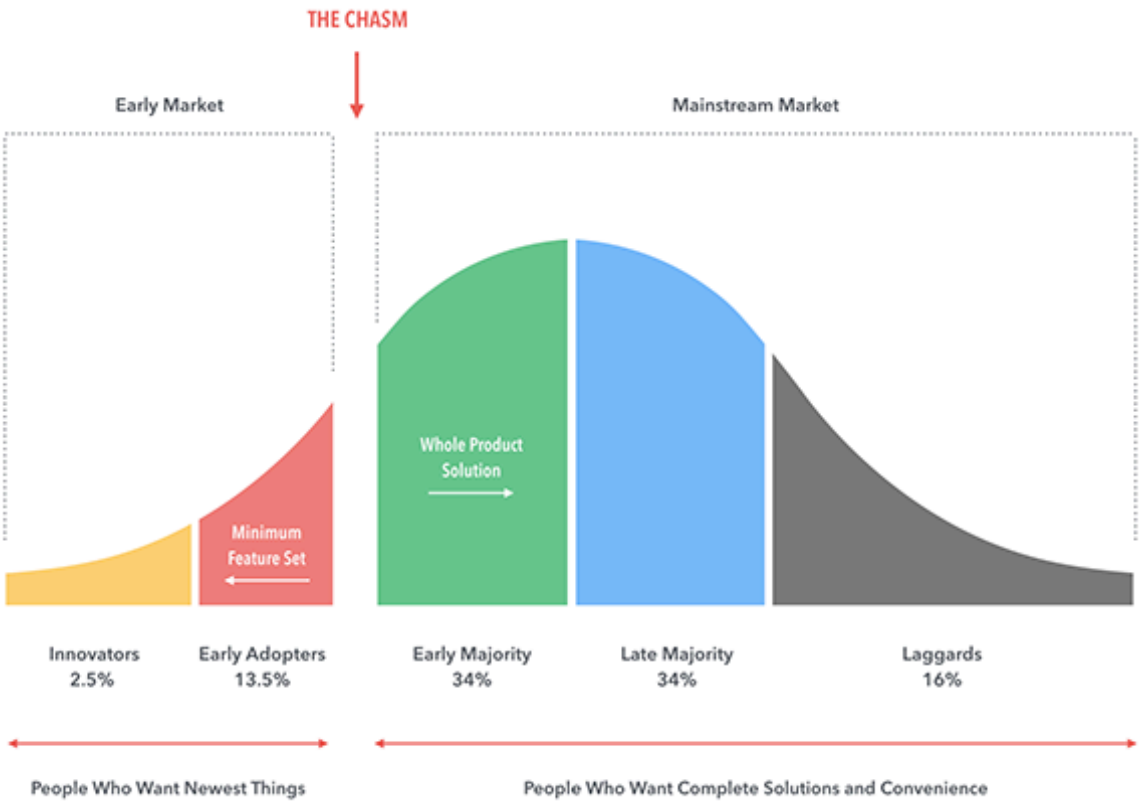


Figura 6.2.1 - Gráfica de difusión de la innovación

Obviamente, un aspecto importante: el ego. A un primer adoptante le encanta ser protagonista, que se hable de él y que se le reconozca su aporte. Contar con un primer adoptante no solo significa aprovechar su potencial, sino distinguir sus méritos.

La opinión de los primeros adoptantes suele ser cualificada, cosa que puede llevarles a ser excelentes prescriptores. El ego bien canalizado puede hacerles tener un gran nivel de influencia en el siguiente grupo de consumidores: Consumidores precozes.
La búsqueda de primeros adoptantes para acompañar en el camino y mejorar las propuestas de productos y/o servicios no es nada sencilla, pero tampoco es imposible. Ahora bien, si aparecen hay que darles importancia, pues su valor puede ser inimaginable.
Algo muy importante es que los primeros adoptantes tienen preferencia por los deseos, lo que significa que no piensan tanto en las necesidades, o al menos, no son su principal motivación. Son un segmento clave dentro de una estrategia que apueste por la innovación como camino a recorrer en el largo plazo.

El resultado de este paso es la definición del MVP.

Paso 3: Prototipar MVP

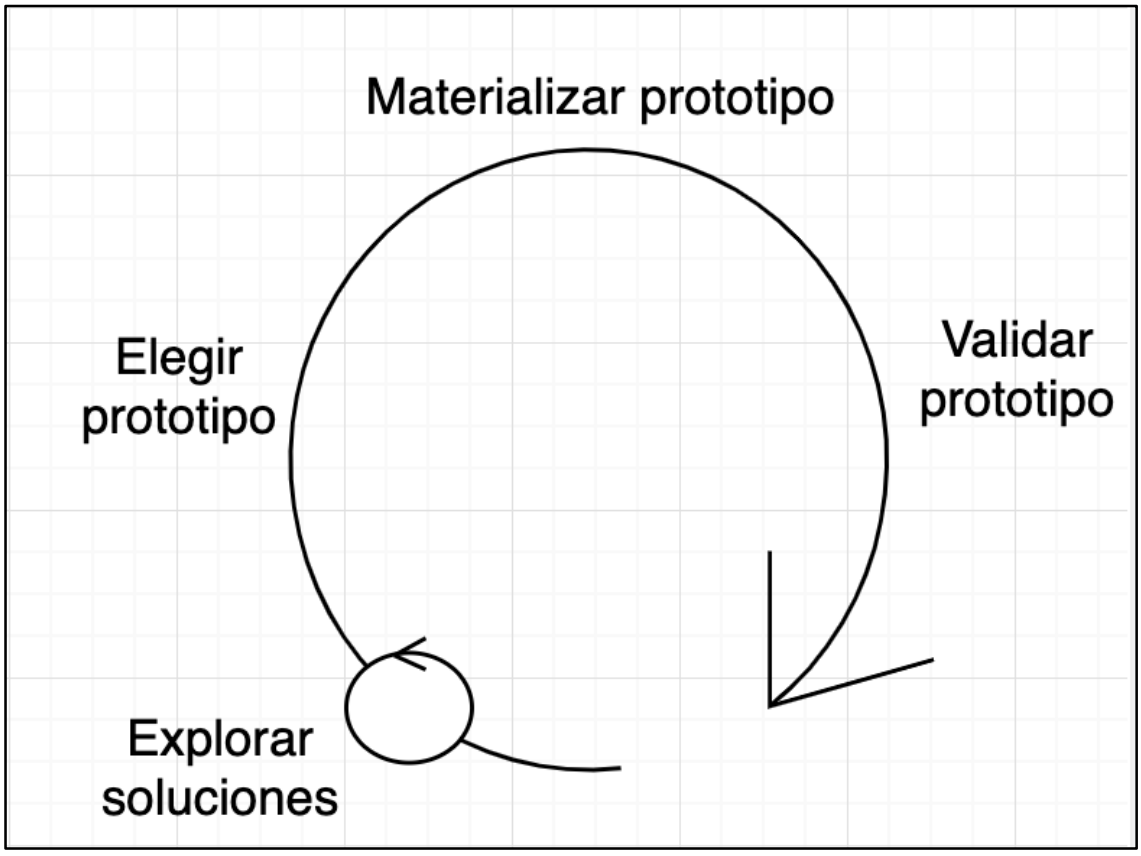


Figura 6.3 : Paso 3 del marco de trabajo

El equipo de diseño es responsable de abordar el problema desde el punto de vista del usuario. Explorar posibles soluciones y presentarlas al equipo para luego decidir cuál és la solución elegida para materializarla y luego construirla en el siguiente paso.

En este paso debemos materializar las ideas para llevarlas al mundo digital.
Un prototipo nos dará la oportunidad de crear un modelo de trabajo casi real de las soluciones que se propuso en la fase de investigación y descubrimiento (paso 1 y 2). Prototipar es emocionante porque la idea realmente comienza a cobrar vida. Las partes interesadas (stakeholders en inglés) y otras personas no técnicas en el equipo también pueden emocionarse durante esta etapa porque los prototipos son más visuales y representan el producto final más de cerca. Durante la etapa de creación de prototipos, lo más probable es que se empiecen a diseñar algunos elementos de diseño visual, se crearán algunos iconos y tal vez incluso una guía de estilo o maqueta. Las maquetas también juegan un papel importante en la etapa de creación de prototipos, especialmente cuando se trata de dar una presentación de UX. La ventaja de crear un prototipo es que se comprenderá mejor cómo será el producto final. Idealmente, se pueden probar los prototipos con los usuarios para aprender y refinar la solución. Pero a veces este punto es complejo de llevarlo a cabo en proyectos reales, por lo que la validación e iteración de validación puede llevarse a cabo internamente en el equipo.

Tener presente el lema “Build the right thing” [14] [21]. En español, “construye el producto correcto”.

El resultado de este paso es el prototipo del MVP.

Paso 4: Implementar MVP

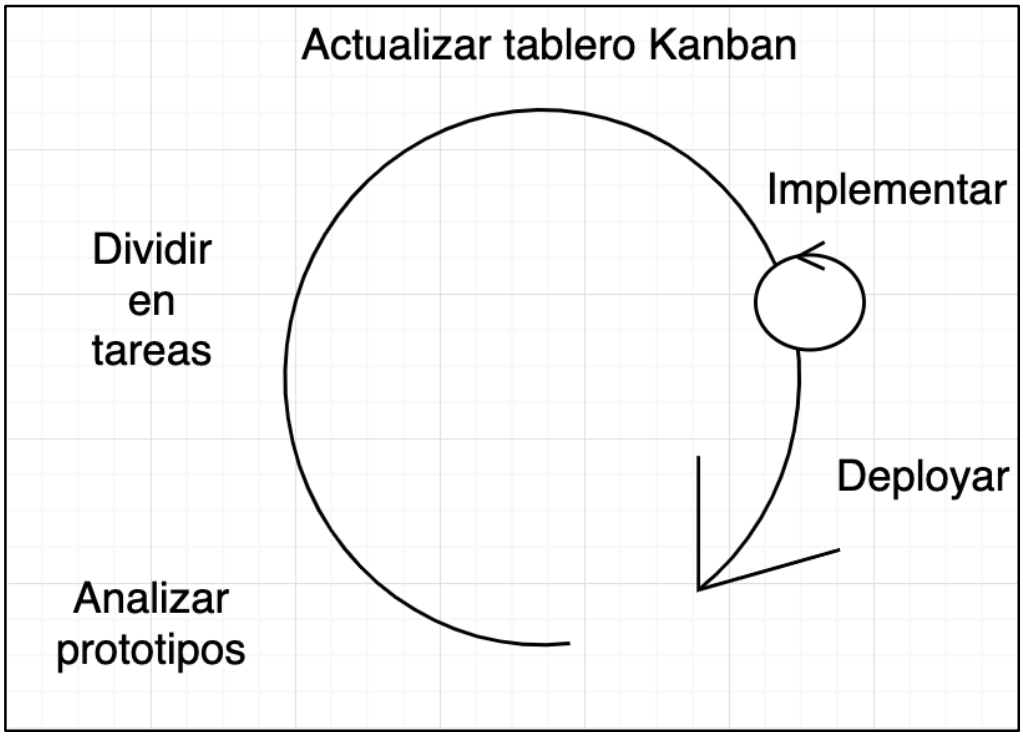


Figura 6.4: Paso 4 del marco de trabajo

Una vez validados los prototipos se pone al equipo de desarrollo a trabajar para construir el MVP. Preparamos los tableros Kanban con las divisiones de tareas necesarias. Como se dijo anteriormente, vamos a utilizar ReactJS para la implementación. Aquí las ventajas de este framework afloran. Como está directamente relacionado a construir componentes, al visualizar los prototipos, lo primero que se debe analizar es:

- ¿Qué componentes podemos identificar en el prototipo?
- ¿Cual es la complejidad de esos componentes?
- ¿Los componentes del prototipo, son tan complejos que pueden dividirse en tareas ?

De esta manera iremos armando nuestro tablero Kanban con las tareas resultantes.
Es importante la coordinación de la implementación con el mantenimiento de los tableros para poder en todo momento saber qué se está haciendo, que está pendiente y qué está hecho. Por lo tanto el equipo de desarrollo es responsable de ir cambiando el estado de las tareas y comunicando al equipo de producto sobre esos cambios.
En este paso se paraleliza varias tareas. Por un lado el equipo de diseño está un paso adelantado del equipo de desarrollo. Es decir, por conveniencia e idealmente, cuando equipo de desarrollo se encuentre en una iteración n, el equipo de diseño deberá estar prototipando y materializando la iteración n+1. Recordamos que, según Lean UX la iteración tanto de diseño como de implementación, pueden paralelizar y lo más común es que eso suceda de una manera en particular. El equipo de diseño debe anticiparse y estar por delante del equipo de desarrollo, y tener los prototipos analizados, validados y listos antes de que el equipo de desarrollo tenga la disponibilidad para implementarlos. Esto es para mantener la agilidad y la velocidad de desarrollo lo más alto posible.

Principalmente en este paso es necesario implementar todas las tareas que se consideren necesarias para que el MVP se materialize.

Tener presente el lema “Build the thing right” [14] [21]. En español, “construye el producto correctamente”.

El resultado de este paso es el MVP construido y accesible al usuario.

Paso 5: Medir

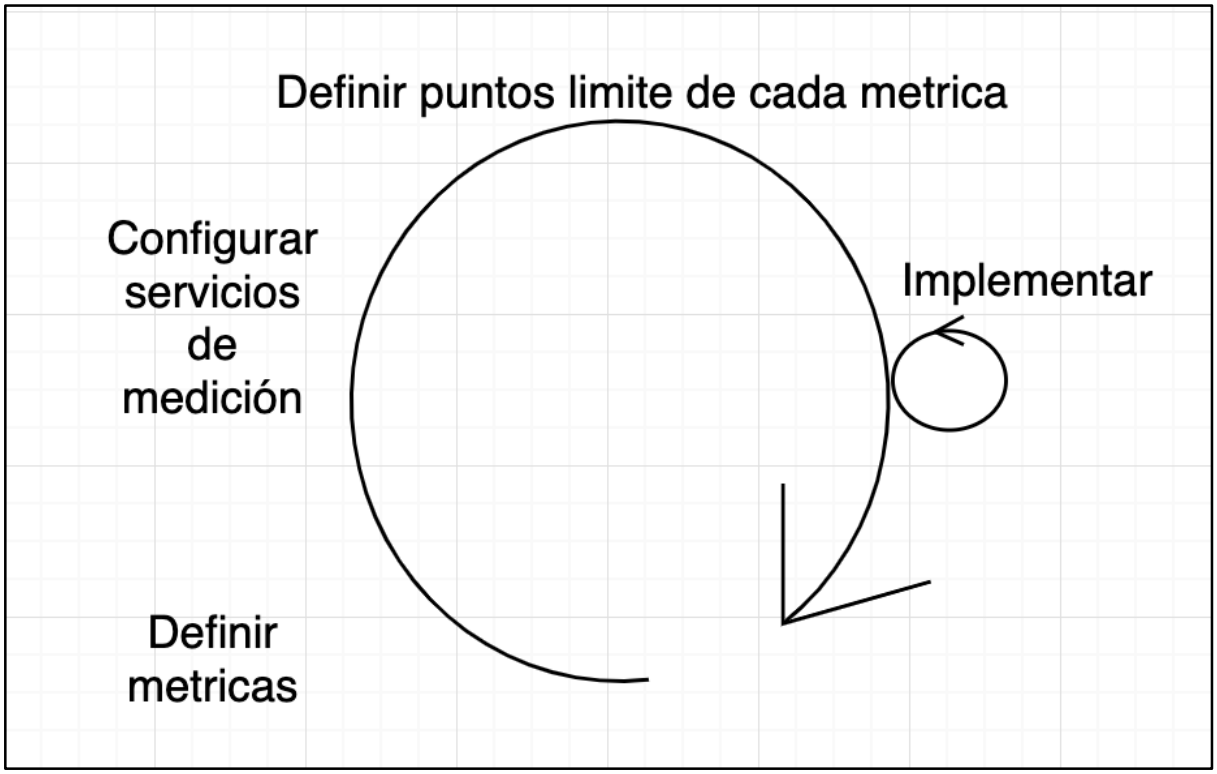


Figura 6.5 : Paso 5 del marco de trabajo

Este paso consiste mayormente en definir cuáles son las métricas cualitativas y cuantitativas que debemos medir, para poder obtener las respuestas necesarias para decidir y validar la hipótesis definida en el paso 1. Medir ayuda a entender por qué los usuarios interactúan con el producto de la manera en que lo hacen. Cómo así también permitirá saber si los usuarios necesitan o quieren lo que se implementó. En otras palabras permitirá validar la hipótesis.
Existen muchas maneras de medir los resultados de una iteración. Todas ellas dependen del servicio que se va a utilizar para medir. Existen cientos de servicios que permiten guardar información relevante de los usuarios. Estos servicios por lo general consiguen la información relevante que necesitaremos más adelante para conocer las características principales de los usuarios que les interesa nuestro servicio. La información puede ser información demográfica, de intereses, gustos personales, de seguimiento, de marketing, de fuentes por los que llegan al sitio, eventos que realicen navegando en el sitio, etc. Un ejemplo de este tipo de servicio es Google Analytics [28]. Es un servicio extensamente utilizado en los sitios web de servicios, ya que dada su versatilidad en la configuración, permite definir muchos niveles diferentes de granularidad en la información a guardar. También permite observar de manera adecuada toda esa información en rangos de tiempo.

[28] Google Analytics
https://analytics.google.com/analytics/web/

Una vez configurados los servicios o herramientas que se van a usar para medir, debemos definir para cada métrica, cuáles son los valores límites tolerables para cada caso. Es decir, los valores máximos y mínimos que estamos dispuestos a tolerar de cada métrica, que tienen un significado positivo/negativo para la métrica.
Por último, como a veces para medir es necesario escribir código, según el servicio que se use, hay que implementar el guardado de la métrica.

El resultado de este paso es la medición de diferentes variables para determinar si nuestra hipótesis del paso 1 es correcta.

Paso 6: Analizar, validar y aprender.

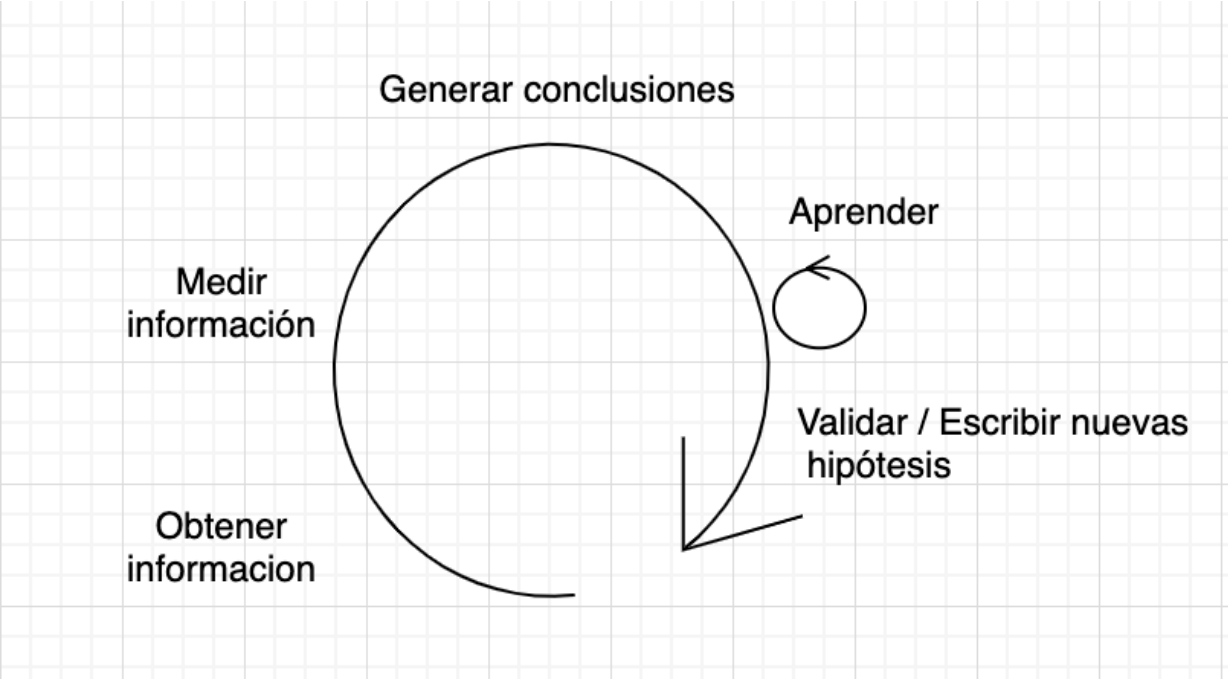


Figura 6.6: Paso 6 del marco de trabajo

La última etapa de este marco consiste en analizar la información recolectada con las servicios y/o herramientas utilizadas en el paso 5, utilizando alguna ventana de tiempo hacia el pasado. Por ejemplo 7 días atrás, luego de la puesta en producción del MVP. De esa manera se contará con información suficiente para sacar alguna conclusión y accionar en base a esa conclusión. Esta etapa es una etapa que es paralela y vertical al resto de las etapas, porque mientras se va pasando del paso 1 al paso 5, siempre se está analizando la información que se va recolectando de iteraciones anteriores. A medida que se van definiendo nuevas hipótesis, la idea es saber qué variables definen la veracidad de la hipótesis planteada, y a medida que se van obteniendo los datos, ir decidiendo si la hipótesis es verdadera o falsa. De esta manera se puede controlar en qué lugar del proyecto es necesario enfocar el equipo y en qué parte es necesario más o menos esfuerzo según lo que se vaya obteniendo como resultado del análisis

de la medición. También puede que el resultado del análisis de la información sea otra hipótesis más acotada o mejor definida, o más amplia porque la información obtenida no es suficiente para arriesgarse a tomar una decisión. Todas estas decisiones de validación por lo general las toman las personas que están encargadas de darle dirección al proyecto junto con el equipo de desarrollo y diseño. Porque es necesario contar con la posibilidad de debatir si los resultados son válidos, si tienen sentido, y si en caso de tener que definir una nueva hipótesis, esta sea viable e implementable.

El resultado de este paso es el aprendizaje sobre lo que se implementó para generar nuevas hipótesis más refinadas, o mejorar las existentes.

Capítulo 7

Implementación de ejemplo con marco de trabajo

Vamos a crear un proyecto a modo de ejemplo, para aplicar los pasos del marco de trabajo y ver cómo funciona el flujo de trabajo entre los equipos multidisciplinarios de una empresa. Este ejemplo está limitado por el alcance definido en el capítulo 6.

Paso 1: Definir hipótesis

Como dijimos, vamos a crear un proyecto desde cero, y para eso necesitamos una idea. A modo de ejemplo se va a utilizar la siguiente idea de proyecto:

“Vamos a crear una aplicación que permita a usuarios contratar un servicio de lavado de vehiculos a domicilio.”

Desde un principio, aplicaremos la filosofía Lean y no asumimos nada sobre la aplicación a implementar. No sabemos nada de los usuarios de la aplicación. Solo tenemos nuestra idea, nuestra primer hipótesis que debemos implementar para validar si lo que pensamos es correcto o no y crear nuestro primer MVP.
Como vamos a estar enfocados en software de consumo, va a haber usuarios y va a ser un servicio accedido a través de internet.
Entonces nuestra idea en forma de suposición sería de la siguiente manera:

“Creemos que las personas que tienen vehículos tienen una necesidad de lavar su auto y no pueden hacerlo por falta de tiempo o por desgano de moverse de su casa.”

La suposición nos lleva a establecer nuestra primer hipótesis inicial de la siguiente manera:

“Creemos que la contratación de un servicio de lavado de automóviles a través de una aplicación se logrará si aquellas personas que no dispongan de tiempo para hacerlo por ellos mismos, utilicen el servicio para lavar su auto en su domicilio.”

Esta es nuestra primer hipótesis y sobre la cual vamos a trabajar y crear nuestro Producto Mínimo Viable (MVP).

Paso 2: Definir MVP

Ya tenemos nuestra primer hipótesis escrita. Por lo cual, el siguiente paso será validar nuestra hipótesis.
Utilizaremos un MVP para validar nuestra primer hipótesis para saber si los usuarios realmente quieren nuestro servicio.
Comenzamos ideando la solución para establecer las tareas necesarias y luego vamos creando nuestro primer tablero Kanban con las primeras tareas y así vamos dando forma concreta al proyecto:

Pendiente	En desarrollo	Terminado
1. Diseñar página de destino		
2. Crear base de datos para guardar los correos electrónicos de los early adopters.		
3. Startup del proyecto. Repositorios de código. Estructura principal del proyecto (carpetas y archivos).		
4. Armar lógica backend para recibir formulario con el correo del usuario y guardarlo en la base de datos.		
5. Construir página web de destino.		
6. Preparar servicio de hosting para alojar página de destino.		

Necesitamos poner manos a la obra, poner el equipo en funcionamiento para diseñar y construir el primer MVP y conseguir esos primeros adoptantes.
Lo primero que vamos a hacer es diseñar una página de bienvenida, que contendrá solamente una breve descripción de lo que resuelve el servicio y un campo de texto para que el usuario deje su email si está interesado en recibir más información cuando el software esté terminado y en funcionamiento. De esta manera podemos validar que existan personas interesadas en nuestro servicio **sin empezar a hacerlo**.

Podemos reconocer de inmediato, que empezar a construir el servicio antes de validarlo, puede llegar a generar mucho desperdicio que se transformará en perdida de dinero y tiempo. Aquí es donde la filosofía Lean nos ayuda a tomar el camino correcto.
Por otro lado, la experiencia nos permite ir agregando al tablero Kanban, tareas relacionadas a la gestión del código y a las tarea de infraestructura necesaria para soportar la puesta en produccción del MVP, mas allá de que aún ni siquiera esté construido.

Paso 3 : Prototipar MVP

Como estamos usando un modelo ágil, luego de agregar nuestras primeras tareas Kanban, estamos listos para poner al equipo de diseño a trabajar junto con el equipo de desarrollo. Aquí es donde los conceptos de Lean UX afloran para hacer que todo fluya hacia el resultado. Como se definió en este paso, el equipo de diseño es responsable de abordar el problema desde el punto de vista del usuario. Explorar soluciones para luego materializarlas.
Veamos entonces cómo está nuestro tablero kanban en esta etapa:

Pendiente	En desarrollo	Terminado
	1. Diseñar página de destino	
	2. Crear base de datos para guardar los correos electrónicos de los early adopters.	
3. Startup del proyecto. Repositorios de código. Estructura principal del proyecto (carpetas y archivos). Tareas no funcionales.		
4. Armar lógica backend para recibir formulario con el correo del usuario y guardarlo en la base de datos.		
5. Construir página web de destino.		
	6. Preparar servicio de	

	hosting para alojar página de destino.	
7. Diseñar email de respuesta al usuario cuando se registra.		

Claramente mientras el equipo de diseño diseña nuestra primer página de destino, el equipo de tecnología puede ir avanzando en otras tareas aisladas pero que servirán de soporte, hasta que la tarea 1 esté terminada.

Se agregó la tarea 7, que implica diseñar la respuesta al usuario, informando que el servicio aún no está listo y que le avisaremos y le iremos enviando información acerca del estado del proyecto a medida que se vaya construyendo. Esta es nuestra primer validación. Es decir, los usuarios dejan sus datos y se registran, y nosotros podemos volver a contactarlos por email para preguntarles si estarían dispuestos a esperar que el servicio esté completamente en línea por ejemplo. Profundizaremos sobre la etapa de validación / crecimiento más adelante.

Solo a modo de ejemplo este prototipo va a ser el resultado de esta etapa, la página de bienvenida:

Figura 7.2 - Primer prototipo de MVP

Con nuestra página diseñada, el equipo de desarrollo puede comenzar a construirla y hacerla funcional. Para ello es necesario que mientras el equipo trabajaba en el diseño, el equipo técnico haya preparado los ambientes y recursos necesarios para

guardar los datos que los usuarios envían a través del formulario, es decir nuestra base de datos (tarea 2) , y también hayan preparado los ambientes de hosting para alojar nuestra página de destino (tarea 6). Por último, el equipo de desarrollo tiene que comenzar a armar el ambiente de desarrollo haciendo el startup o setup del código principal, con la estructura de archivos y carpetas, creación del repositorio de código para poder tenerlo en la nube y compartirlo con el resto del equipo, armar archivos ejecutables para poder subir la aplicación al servicio hosting, etc.. Es una tarea que involucra todas las tareas no funcionales pero si necesarias para que el MVP cobre vida (tarea 3).
Para simplificar el trabajo, omitiremos la implementación de estas tres tareas: 2, 3 y 6 Las asumimos como terminadas en el siguiente paso.

Paso 4: Implementar MVP

Con el diseño terminado y la infraestructura lista para poner en línea nuestro primer MVP, solo queda comenzar con la implementación. El equipo de desarrollo tiene en sus manos el prototipo seleccionado para materializarlo. Es momento de analizarlo y dividirlo en tareas. De esa manera actualizamos el tablero kanban con las tareas resultantes. Entonces en esta etapa, nuestro tablero Kanban se ve de la siguiente manera:

Pendiente	En desarrollo	Terminado
		1. Diseñar página de destino
		2. Crear base de datos para guardar los correos electrónicos de los early adopters.
		3. Startup del proyecto. Repositorios de código. Estructura principal del proyecto (carpetas y archivos).
	4. Armar lógica backend para recibir formulario con el correo del usuario y guardarlo en la base de datos.	
	5. Construir página web de destino.	

		6. Preparar servicio de hosting para alojar página de destino.
	7. Diseñar email de respuesta al usuario cuando se registra.	
8. Lean UX: Check: Preparar analitica del sitio. Definir variables a medir.		
9. Lean UX: Check: Preparar reuniones con clientes para preguntar acerca de si estarían dispuestos a pagar por el servicio.		

Nuestro tablero kanban sigue de la misma manera. Este paso es para poder dividir la tarea 5 en tareas más pequeñas. Luego de su análisis, el equipo de desarrollo identifica posibles componentes reutilizables en el diseño del MVP, como así también se identifica lógica de negocio vinculada al guardado del email y nombre del usuario que solicita acceso a la aplicación.
En este trabajo, utilizaremos React para escribir todas las soluciones.

Entonces descomponemos la tarea 5 en las siguientes subtareas:

Tarea 5. Construir pagina web de destino.

1. Identificar a simple vista, qué librerías de React se van a utilizar para construir la arquitectura de esta aplicación.
2. Definir estructura del proyecto y navegabilidad de la aplicación para ir pensando en la posibilidad de que la aplicación comience a tener otras páginas navegables como login, formulario de registro, panel del usuario, etc.
3. Definir componentización lógica de la aplicación.

Las 3 subtareas 5.1, 5.2 y 5.3, son tareas resultantes del análisis de la aplicación a simple vista.
En 5.1 tendremos que pensar si vamos necesitamos alguna librería que nos facilite la creación de la página en este punto. Por ejemplo algún framework que tenga componentes de UI como Material-ui [29] para React.
Esta librería provee de componentes visuales listos para utilizar con React. Es decir, sobreescribe los componentes que el navegador ya sabe renderizar, como por

[29] https://material-ui.com/

ejemplo un campo de texto de un formulario o un texto de párrafo, y le agrega funcionalidad a esos objetos web básicos utilizando React. De esa manera, en el html en vez de escribir por ejemplo:

<input /> o

Se escribe:

<TextInput /> o <List><ListItem></ListItem></List>

Este paso es necesario realizarlo antes de empezar a escribir código, porque son librerías que definen qué componentes van a ser los componentes principales o root de la aplicación.

En 5.2 hay que comenzar a escribir el primer código que por ejemplo comience a definir la estructura de navegación de la aplicación. Para ello se puede utilizar otra librería de react frecuentemente utilizada que se llama react-router [30]

En 5.3 simplemente analizamos el prototipo identificando "físicamente" los componentes a renderizar en la aplicación. Por ejemplo:

Figura 7.3 - Primer prototipo MVP. Análisis de componentes.

Encontramos 4 componentes visibles. Como se va a utilizar la librería para React material-ui que provee de componentes de UI listos para ser usados, rápidamente

[30] https://reacttraining.com/react-router/

podemos decidir que todo lo que se ve en el prototipo son componentes del estilo web estándar. No existe ningún componente complejo para desarrollar aún, que surja de este prototipo. En este caso son:

1. Componentes de títulos o heading de diferentes tamaños y colores
2. Campos de textos de formulario
3. Botón principal amarillo
4. Texto genérico informativo

A continuación pondremos una parte del código resultante de esta primer página de nuestra aplicación. A modo de ejemplo, hay partes del código que se omiten. Nos enfocamos en escribir el código necesario para mostrar la página como el equipo de diseño prototipó, y mostraremos cómo sería una implementación simple de la lógica para el envío del formulario. Por supuesto, como dijimos, todo utilizando React:

```
import React, { Component } from 'react';
import './App.css';

//importamos los componentes desde la libreria material-ui
import Typography from '@material-ui/core/Typography'
import TextField from '@material-ui/core/TextField';
import Button from '@material-ui/core/Button';


//Estado para almacenar los valores del formulario
interface State {
  nombre: string,
  email: string
}

//Usado solo a modo de ejemplo. Necesario para React
//Contiene propiedades enviadas desde componentes padres
interface Props {

}

//A modo de prueba creamos una clase que va a ser responsable de guardar el
//usuario en la base de datos. Pero no la implementamos
interface Backend {
  guardarNuevoUsuario: (nombre: string, email: string) => void
}

class Backend implements Backend {
  //no se implementa por ahora
}

//Clase Typescript extiende desde React.Component
class App extends Component<Props, State> {
```

```
  private backend: Backend;

  constructor(props: Props) {
    super(props);
    //inicializamos el estado
    this.state = {
      nombre: '',
      email: ''
    }
    this.backend = new Backend();
  }

  //Envia el formulario al backend para guardarlo
  private enviarFormulario() {
    this.backend.guardarNuevoUsuario(this.state.nombre, this.state.email);
  }
  //Renderización del componente
  //El lenguaje utilizado aquí es JSX
  //Mezcla html y javascript al mismo tiempo para no separar la UI
  //de la lógica como vimos en el capitulo 5
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <Typography variant="h1">Lavadero online</Typography>
          <Typography variant="h2">Lavamos su auto en su casa</Typography>
          <Typography variant="h2">Registrese ahora</Typography>
          <form noValidate autoComplete="off">
            <TextField
              id="nombre"
              label="Nombre"
              value={this.state.nombre}
            />
            <TextField
              id="email"
              label="Email"
              value={this.state.email}
            />

            <Button onClick={this.enviarFormulario} variant="contained">
              Quiero 20% de descuento ahora
            </Button>

            <p>disclaimer</p>
          </form>
        </header>
      </div>
    );
  }
}
```

```
export default App;
```

De esta manera, con este pseudocódigo, concluimos con la implementación del primer MVP.
En este punto es un buen lugar para una comparación implementando este MVP sin ningún framework o con otros frameworks que no sean orientado a componentes. En general, tendríamos que por un lado haber escrito nuestro código html supongamos en un archivo .html, el código javascript en un archivo .js de javascript y en un archivo .css el código necesario para aplicar los estilos visuales de la página. Una de las principales ventajas de React es que a través del lenguaje JSX, permite escribir el código html junto con el código javascript. Entonces de esa manera cada componente define cómo se muestra a través del código html que devuelve en su método render().
Para finalizar este paso, una vez escrito el código, solo queda empaquetar el código y subir nuestro nuevo MVP al servicio de hosting.

Paso 5: Medir

Una vez hecho el deploy de la aplicación nos queda comenzar a medir para luego analizar y validar si lo que dijimos en nuestra hipótesis del paso 1 es verdadero o falso.
En esta etapa nuestro tablero Kanban se encuentra de esta manera:

Pendiente	En desarrollo	Terminado
		2. Diseñar página de destino
		2. Crear base de datos para guardar los correos electrónicos de los early adopters.
		3. Startup del proyecto. Repositorios de código. Estructura principal del proyecto (carpetas y archivos).
		4. Armar lógica backend para recibir formulario con el correo del usuario y

		guardarlo en la base de datos.
		5. Construir página web de destino.
		6. Preparar servicio de hosting para alojar página de destino.
		7. Diseñar email de respuesta al usuario cuando se registra.
	8. Lean UX: Check: Preparar analitica del sitio. Definir variables a medir.	
	9. Lean UX: Check: Preparar reuniones con clientes para preguntar acerca de si estarían dispuestos a pagar por el servicio.	

La tarea 8 es una tarea que requiere implementación del equipo de desarrollo, para conectar nuestra aplicación con los servicios que van a guardar la información de seguimiento de los usuarios que naveguen nuestro sitio, usuarios que completen el formulario y envíe sus datos. También es necesario definir cuales son las variables de interés o la información que es de nuestro interés y relevante para la prueba de este MVP.
Entonces para MEDIR, la tarea 8 va a ser configurar y probar la integración con alguna herramienta de medición como Google Analytics, como así también definir qué es lo que se va a medir.
En la tarea 9, no necesitaremos implementación del equipo de desarrollo ya que la idea es contactar a aquellos usuarios que han decidido completar y enviar el formulario. Principalmente lo que queremos saber es si estos usuarios estarían dispuestos a esperar a que el servicio esté finalizado y si estarían dispuestos a pagar por el mismo. Con estos datos podemos empezar a validar nuestro MVP con información real, y no con suposiciones. Incluso antes de seguir gastando tiempo y dinero en quizás una idea que no sea de interés a nadie.
Al finalizar este paso 5, lo que tiene que pasar es que ya deberíamos tener el MVP funcionando y midiendo lo que se haya definido como de interés para medir. Por esa razón luego de este paso, ya podemos comenzar con una nueva iteración comenzando desde el paso 1 nuevamente, validando una hipótesis nueva.

Para nuestro ejemplo, puntualmente nos interesa medir las siguientes métricas cuantitativas:

- Cantidad total de usuarios que visitan la página.
- Cantidad total de usuarios que dejan sus datos (conversión).
- Fuente o medio por el que conocen nuestro servicio.

En cuanto a métricas cualitativas, nos interesa medir:

- Información demográfica y geográfica de los usuarios.
- Nivel de Interés real por el servicio (encuestas)

Paso 6: Analizar mediciones, validar MVP y aprender.

Es momento de revisar los valores de las métricas para sacar conclusiones. Para nuestro ejemplo tenemos varias métricas por revisar. Como es un ejemplo teórico, vamos a simular un escenario favorable y uno no favorable en cuanto a los valores de las métricas, para ver qué acciones tomaremos en cada caso.
Para el caso de valores favorables, podemos suponer los siguientes valores:

Cuantitativas	**Cualitativas**	**Resultado**
Total de visitas		1000
Cantidad de usuarios registrados / conversión		700 / 70%
Fuente / medio		Google / direct
	Info demográfica / geográfica	Hombres y mujeres / Argentina
	Nivel de interés (encuestas)	60%

Ejemplo de métricas en un escenario favorable

Luego de analizar los valores de las métricas elegidas en este escenario, podemos sacar las siguientes conclusiones:

1. Que el nivel de conversión (total de visitas / usuarios registrados) es muy alto y eso nos puede dar la idea de que existen personas interesadas en nuestro servicio, y por ese motivo se registran.

2. Podemos empezar a definir un perfil para los usuarios de nuestro servicio, analizando la información de fuente/medio (es decir por donde llegan a nuestro sitio), información demográfica y geográfica de los usuarios.
3. Por último del resultado de las encuestas podemos deducir que el 60% de los usuarios que dejaron su correo, estarían interesados en esperar a que el servicio esté terminado para usarlo.

Claramente este es un escenario sumamente favorable, que nos permite validar la hipótesis inicial y a su vez, mejorar nuestra hipótesis inicial, como así también crear nuevas hipótesis.

Capítulo 8

Comparación con otros marcos de trabajo

Introducción

En la actualidad existen muchas metodologías de desarrollo de software ágil. Cada una con sus ventajas y desventajas. Es decisión de cada empresa adoptar una de esas metodologías para desarrollar su software.
Algunas de las metodologías más usadas y conocidas son:

Programación Extrema (XP): también conocida como XP, la Programación Extrema es un tipo de desarrollo de software destinado a mejorar la calidad y la capacidad de respuesta a los requisitos cambiantes de los clientes. Los principios de XP incluyen retroalimentación, asumiendo simplicidad y abarcando el cambio.

Desarrollo impulsado por características (FDD): este proceso de desarrollo de software iterativo e incremental combina las mejores prácticas de la industria en un solo enfoque. Hay cinco actividades básicas en FDD: desarrollar un modelo general, crear una lista de características, planear por característica, diseñar por característica y construir por característica.

Desarrollo del sistema adaptativo (ASD): El desarrollo del sistema adaptativo representa la idea de que los proyectos siempre deben estar en un estado de adaptación continua. ASD tiene un ciclo de tres series repetitivas: especular, colaborar y aprender.

Método de desarrollo de sistemas dinámicos (DSDM): este marco de entrega de proyectos Agile se utiliza para desarrollar software y soluciones que no sean de TI. Aborda las fallas comunes de los proyectos de TI, como sobrepasar el presupuesto, los plazos faltantes y la falta de participación del usuario. Los ocho principios de DSDM son: centrarse en la necesidad del negocio, entregar a tiempo, colaborar, nunca comprometer la calidad, construir de manera incremental a partir de bases firmes, desarrollar de forma iterativa, comunicarse de manera continua y clara, y demostrar control.

Crystal Clear: Crystal Clear es parte de la familia de metodologías Crystal. Se puede usar con equipos de seis a ocho desarrolladores y se enfoca en las personas, no en los procesos o artefactos. Crystal Clear requiere lo siguiente: entrega frecuente de código utilizable a los usuarios, mejora reflexiva y comunicación osmótica, preferiblemente por ubicación conjunta.

Scrum: Scrum es una de las formas más populares de implementar Agile. Es un modelo de software iterativo que sigue un conjunto de roles, responsabilidades y reuniones que nunca cambian. Los sprints, que suelen durar de una a dos semanas, permiten que el equipo entregue el software de forma regular.

Para mayor claridad, el marco de desarrollo propuesto en este trabajo, reune las caracteristicas de los modelos de Kanban y Lean Software Development.

Claramente, todos los modelos ágiles son muy similares, y no podemos realizar una comparación con cada uno porque no es el objetivo de este trabajo. Vamos a seleccionar uno de ellos para realizar una comparación y ver las ventajas y desventajas de este marco y el modelo seleccionado. El modelo que vamos usar para comparar es SCRUM.

¿Qué es Scrum?

Scrum es un subconjunto de Agile y uno de los marcos de procesos más populares para la implementación de Agile. Es un modelo de desarrollo de software iterativo utilizado para administrar software complejo y desarrollo de productos. Las iteraciones de longitud fija, llamadas sprints que duran de una a dos semanas, permiten al equipo enviar software en una cadencia regular. Al final de cada sprint, las partes interesadas y los miembros del equipo se reúnen para planificar los próximos pasos.

Scrum sigue un conjunto de roles, responsabilidades y reuniones que nunca cambian. Por ejemplo, Scrum requiere cuatro reuniones que brinden estructura a cada sprint: planificación de sprint, stand-up diario, demostración de sprint y retrospectiva de sprint. Durante cada sprint, el equipo utilizará artefactos visuales como tableros de tareas o gráficos para mostrar el progreso y recibir retroalimentación incremental.

Jeff Sutherland creó el proceso Scrum en 1993, tomando el término "Scrum" de una analogía en un estudio de 1986 realizado por Takeuchi y Nonaka publicado en Harvard Business Review. En el estudio, Takeuchi y Nonaka comparan equipos multifuncionales de alto rendimiento con la formación Scrum utilizada por los equipos de Rugby. El contexto original para esto fue la industria de fabricación, pero Sutherland, junto con John Scumniotales y Jeff McKenna, adaptaron el modelo para el desarrollo de software. [23]

Comparación de Scrum con el marco de trabajo

Mientras que Scrum ofrece algunos beneficios concretos, también tiene algunas desventajas. Scrum requiere un alto nivel de experiencia y compromiso por parte del equipo y los proyectos pueden correr el riesgo de que se produzcan errores.

A continuación enumeramos los puntos de comparación:

- Riesgo de que se produzcan cambios en el alcance: Todos los proyectos pueden experimentar un aumento en el alcance debido a la falta de una fecha final específica. Sin fecha de finalización, las partes interesadas pueden verse tentadas a seguir solicitando funcionalidad adicional. Este punto es super crítico, y es normal que ocurra porque en SCRUM el alcance se define al comienzo de la iteración y no se aceptan nuevos requerimientos o modificaciones una vez que se está produciendo código. Entonces, si el cliente cambia por necesidad los requerimientos, la situación es crítica. Se tiene que dejar de hacer o se hace y luego se tira, ya que el cliente manifiesta en la presentación de la iteración, que el requerimiento ha cambiado. Claramente esto es un problema en SCRUM.
  En el marco de trabajo, si el cliente tiene una nueva necesidad o cambiar algo que se está haciendo, esto se resume solamente a agregar una tarea nueva a nuestro tablero KANBAN, o crear una nueva hipótesis luego de analizar los resultados de la misma. Porque el flujo es pensar - hacer - medir - analizar. Entonces se supone que nuestro cliente no va a tener datos para tomar nuevas decisiones hasta que no se haya implementado la hipótesis original.

- El equipo requiere experiencia y compromiso: con roles y responsabilidades definidos, el equipo debe estar familiarizado con los principios de Scrum para tener éxito. Debido a que no hay roles definidos en el Equipo Scrum (todos lo hacen todo), se requiere que los miembros del equipo tengan experiencia técnica. El equipo también debe comprometerse con las reuniones diarias de Scrum y permanecer en el equipo durante la duración del proyecto.
  En el marco de trabajo, cada persona que integra el grupo de trabajo cumple su rol natural. No existen responsabilidades abstractas ni reuniones mágicas que cumplir. Todo se coordina mirando el tablero, con comunicación natural y con reuniones diarias / semanales para coordinar qué se está haciendo, qué hay para hacer nuevo o que se hizo. Es poner en palabras lo que se ve en el tablero todo el tiempo. Y a su vez ir modificando el tablero a medida que ocurren cambios de estado en las tareas existentes, o se agregan/eliminan tareas.

- El Scrum Master incorrecto puede arruinar todo: Scrum Master es muy diferente de un administrador de proyectos. El Scrum Master no tiene autoridad sobre el equipo; él o ella debe confiar en el equipo que están administrando y nunca decirles qué hacer. Si el Scrum Master intenta controlar el equipo, el proyecto fallará.
  Scrum propone este rol como necesario para llevar adelante la metodología. En el marco de trabajo no necesitamos ningún rol coordinador o especial.

- Las tareas mal definidas pueden dar lugar a inexactitudes: los costos y los plazos de los proyectos no serán precisos si las tareas no están bien definidas.

Si los objetivos iniciales no están claros, la planificación se vuelve difícil y los sprints pueden llevar más tiempo que el estimado originalmente. Este es el problema principal de SCRUM, y es el problema que este marco de trabajo propone solucionar desde sus entrañas. Las tareas mal definidas ocurren por varios motivos:

- Si bien tanto en SCRUM como en el marco de trabajo se pueden crear tareas (hipótesis) desde suposiciones, el problema de SCRUM es que no hay una estructura para validar si la suposición era correcta o no. En el marco de trabajo esto es el núcleo. Suponer/pensar, hacer, medir y validar. Por lo tanto estamos constantemente validando nuestras suposiciones para no generar desperdicios y tener al equipo trabajando en tareas que tengan sentido para el negocio.
- En la práctica, en SCRUM, las tareas terminan siendo definidas más por el equipo técnico, que por los analistas o líderes de proyecto. Esto ocurre porque, como es necesario hacer un cúmulo de tareas por sprint, y esa cantidad no es negociable ya que depende de la velocidad de desarrollo del equipo, las tareas terminan siendo “refinadas/modificadas” para que entren en el sprint. Esto genera que la tarea finalmente termine siendo refinada sin sentido quedando incompleta, y no satisface la necesidad real del cliente (interno o externo).

Capítulo 9

Conclusiones

Este trabajo promueve el desarrollo de metodologías ágiles para el desarrollo de proyectos orientados a servicios, para empresas con equipos multidisciplinarios. Es muy fácil de implementar, no requiere conocimientos previos en ninguna metodología. Solo entender los pasos para que cada integrante del equipo esté enfocado en lo que sabe hacer. Promueve la utilización del pensamiento Lean tanto para el equipo técnico como para el equipo no técnico. Mantiene al equipo en general enfocado en "construir el producto correcto" y en "construir bien", pensando en cómo trabajar para no generar desperdicios y garantizar que el proyecto no se quede sin recursos, ni ideas, por el desgaste que se produce cuando se debe tirar a la basura días o meses de trabajo, por no haber hecho una clara planificación o incepción de requerimientos. El marco de trabajo cubre todas las etapas que ocurren en una empresa, cuando se quiere poner en marcha una idea, hasta la implementación y validación con datos reales de la misma. Cómo podemos ver, no hay una metodología que abarque todas las etapas y a su vez indique cómo implementarla. La mayoría deja a la libre interpretación, sobre cómo llevar a adelante el proyecto usando esas metodologías. Como es el caso de SCRUM. Cuando uno lee la metodología, y se alimenta de los conceptos teóricos que propone la metodología, no encuentra el cómo hacerlo. Siempre se habla del qué hacer, pero no del cómo. Y ahí es donde cada uno, o cada empresa elige lo que le sirve para funcionar. El problema de eso es que cada uno aprende SCRUM como quiere. Si bien uno puede tener los conceptos claros, en la práctica puede haber contaminaciones o desviaciones de lo que la teoría plantea, por el simple hecho de implementarlo y adaptarlo a un contexto o empresa en particular. El marco de trabajo, propone desde una metodología de gestión de proyectos y tareas como Kanban, hasta qué tecnología usar para implementar el proyecto como ReactJS. Todas estas decisiones sobre qué usar para cada caso, están 100% enfocadas en ser lo más rentable posible, que sean fáciles de aprender, y que permitan orientar al equipo a trabajar lo más centrado en el usuario posible. Tanto kanban, como ReactJS permiten eso. Kanban porque es fácil de entender fuera de un contexto no tecnológico. Volviendo al ejemplo de SCRUM, es más complicado explicarle a un usuario o cliente cómo el equipo gestiona los tiempos de los proyectos, las tareas, etc que si se le muestra un tablero Kanban, donde solo ve tareas en 3 columnas con 3 estados posibles, pendiente, en curso, hecho.

Por el lado de ReactJs, permite pasar muy rápidamente de los prototipos de diseño a la implementación, porque está 100% orientado a componentes reutilizables, y de esa manera nos permite poder mostrar más rápido la implementación, para poder validarla con los clientes o los usuarios. Además ReactJs es muy contemporáneo, hecho por Facebook, por lo tanto está en puro crecimiento y cuenta con una comunidad muy

extensa que desarrolla constantemente nuevos componentes o plugins que resuelven problemas específicos, y pueden ser utilizados en cualquier proyecto que los necesite.

Trabajos a futuro

Una línea de trabajo a completar a partir de esta propuesta es, poder plantear un marco flexible para el caso de proyectos para software empresarial y no de consumo. Por ejemplo poder flexibilizar el marco o extenderlo para poder trabajar en un proyecto que considere una etapa de migración de un software viejo a una solución más contemporánea. También podría ser útil extender el trabajo y poder agregarle una etapa más al proceso, que hable específicamente sobre cómo escalar el proyecto mientras se va desarrollando desde el punto de vista estratégico.

Referencias Bibliográficas

[1] React – A JavaScript library for building user interfaces. (n.d.). https://reactjs.org/

[2] Fedosejev, A. (2015). React. js Essentials. Packt Publishing Ltd.

[3] Pete Hunt. June 05, 2013. Why did we build React? – React Blog. https://reactjs.org/blog/2013/06/05/why-react.html

[4] Bill Fisher. Facebook. How was the idea to develop React conceived and how many people worked on developing it and implementing it at Facebook?. https://www.quora.com/How-was-the-idea-to-develop-React-conceived-and-how-many-people-worked-on-developing-it-and-implementing-it-at-Facebook

[5] Chávez, S. B., Martín, A. E., Rodríguez, N. R., Murazzo, M. A., & Valenzuela, A. (2012). Metodología ÁGIL para el desarrollo SaaS. In XIV Workshop de Investigadores en Ciencias de la Computación.

[6] Paul, Gil
What Is 'SaaS' (Software as a Service). http://netforbeginners.about.com/od/s/f/what_is_SaaS_software_as_a_service.htm

[7] Ziff Davis. Definition of: SaaS. PC Magazine Encyclopedia. https://www.pcmag.com/encyclopedia/term/56112/saas

[8] Levinson, Meridith. Software as a Service (SaaS) Definition and Solutions. http://www.cio.com/article/109704/Software_as_a_Service_SaaS_Definition_and_Solutions

[9] Ernie Miller. 2008. The Joy of User-Driven Development. https://ernie.io/2008/03/28/the-joy-of-user-driven-development/

[10] Spyros Psychogios. 2014. User Driven Development: The Lean startup principles implemented in enterprise software development. https://blog.inf.ed.ac.uk/sapm/2014/03/12/user-driven-development/

[11] Carlos Eduardo Solieverez. 2003. EDUCACIÓN TECNOLÓGICA PARA COMPRENDER EL FENÓMENO TECNOLÓGICO. https://cyt-ar.com.ar/cyt-ar/images/4/49/Educaci%C3%B3n_tecnol%C3%B3gica_p_fen%C3%B3meno_tecnol%C3%B3gico.pdf

[12] Pablo Lledo. 2014. Gestión Lean Y Ágil De Proyectos.

[13] Dr Daniel T. Jones. Septiembre 2007. https://web.archive.org/web/20121118100410/http://www.institutolean.org/index.php/acerca/que-es-lean

[14] Marine Barbaroux 2016. Untangling UX part 2: how can one fit UCD within Lean and Agile Dev process? https://www.cambridgeconsultants.com/insights/untangling-ux-part-2-how-can-one-fit-ucd-within-lean-and-agile-dev-process

[15] STEVEN DOUGLAS. Agosto 2018. https://www.justinmind.com/blog/complete-guide-to-lean-ux/

[16] Miguel Angel Alvarez. Octubre 2016. https://desarrolloweb.com/articulos/que-es-react-motivos-uso.html

[17] Manifiesto ágil. 2001. http://agilemanifesto.org/iso/es/manifesto.html

[18] Lean for dummies. 2007. Natalie J. Sayer y Bruce Williams.

[19] Running Lean, second edition. 2012. Ash Maurya.

[20] El método Lean Startup. Eric Ries. https://www.leadersummaries.com/ver-resumen/el-metodo-lean-startup

[21] Ries, Eric (2011). The Lean Startup.

[22] Rogers, Everett (16 August 2003). Diffusion of Innovations, 5th Edition. Simon and Schuster. ISBN 978-0-7432-5823-4.

[23] Comparision agile models - Smarsheets - https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban