



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

**INTEGRACIÓN DEL PROCESO DE DISEÑO DE *USER EXPERIENCE* AL CICLO DEVOPS
CI/CD**

André Mendoza Torres

Asesorado por el Ing. José Manuel Ruiz Juárez

Guatemala, julio de 2023

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

**INTEGRACIÓN DEL PROCESO DE DISEÑO DE *USER EXPERIENCE* AL CICLO DEVOPS
CI/CD**

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

ANDRÉ MENDOZA TORRES

ASESORADO POR EL ING. JOSÉ MANUEL RUIZ JUÁREZ

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, JULIO DE 2023

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA



NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. José Francisco Gómez Rivera (a.i.)
VOCAL II	Ing. Mario Renato Escobedo Martinez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Ing. Kevin Vladimir Cruz Lorente
VOCAL V	Br. Fernando José Paz González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO


DECANA	Inga. Aurelia Anabela Cordova Estrada
EXAMINADOR	Ing. Neftalí de Jesús Calderón Méndez
EXAMINADOR	Ing. Gabriel Alejandro Díaz López
EXAMINADOR	Ing. Herman Igor Véliz Linares
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

INTEGRACIÓN DEL PROCESO DE DISEÑO DE *USER EXPERIENCE* AL CICLO DEVOPS CI/CD

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 18 de mayo del 2022.



André Mendoza Torres

Guatemala, 16 de mayo de 2023

Ingeniero
Carlos Alfredo Azurdia
Coordinador de Privados y Trabajos de Tesis
Escuela de Ingeniería en Ciencias y Sistemas
Facultad de Ingeniería - USAC

Respetable Ingeniero Azurdia:

Por este medio hago de su conocimiento que en mi rol de asesor del trabajo de investigación realizado por el estudiante **ANDRÉ MENDOZA TORRES** con carné **201612154** y **CUI 2903 88317 0101** titulado **“INTEGRACIÓN DEL PROCESO DE DISEÑO DE *USER EXPERIENCE* AL CICLO DEVOPS CI/CD”**, luego de corroborar que el mismo se encuentra finalizado, lo he revisado y doy fé de que el mismo cumple con los objetivos propuestos en el respectivo protocolo, por consiguiente, procedo a la aprobación correspondiente.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,


José Manuel Ruiz Juárez
Ing. en Ciencias y Sistemas
Colegiado No. 7945
Ing. José Manuel Ruiz Juárez
Colegiado No. 7945



Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala 23 de mayo de 2023

Ingeniero
Carlos Gustavo Alonzo
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Alonzo:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **ANDRÉ MENDOZA TORRES** con carné **201612154** y CUI **2903 88317 0101** titulado **“INTEGRACIÓN DEL PROCESO DE DISEÑO DE USER EXPERIENCE AL CICLO DEVOPS CI/CD”**, y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo aprobado.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,



Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA



FACULTAD DE INGENIERÍA

SIST.LNG.DIRECTOR.5.EICCSS.2023

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del Asesor, el visto bueno del Coordinador de área y la aprobación del área de lingüística del trabajo de graduación titulado: **INTEGRACIÓN DEL PROCESO DE DISEÑO DE USER EXPERIENCE AL CICLO DEVOPS CI/CD**, presentado por: **André Mendoza Torres**, procedo con el Aval del mismo, ya que cumple con los requisitos normados por la Facultad de Ingeniería.

“ID Y ENSEÑAD A TODOS”



Ingeniero Carlos Gustavo Alonzo
DIRECTOR
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, julio de 2023

Ingeniería Civil, Ingeniería Mecánica Industrial, Ingeniería Química, Ingeniería Mecánica Eléctrica, -Escuela de Ciencias, Regional de Ingeniería Sanitaria y Recursos Hidráulicos (ERIS), Maestría en Sistemas Mención construcción y Mención Ingeniería Vial. Carreras: Ingeniería Mecánica, Ingeniería Electrónica, Ingeniería en Ciencias y Sistemas, Licenciatura en Matemática, Licenciatura en Física. Centros: de Estudios Superiores de Energía y Minas (CESEM). Guatemala, Ciudad Universitaria, Zona 12, Guatemala, Centroamérica



USAC
TRICENTENARIA
Universidad de San Carlos de Guatemala

Decanato
Facultad e Ingeniería

24189101- 24189102

LNG.DECANATO.OIE.31.2023

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería En Ciencias Y Sistemas, al Trabajo de Graduación titulado: **INTEGRACIÓN DEL PROCESO DE DISEÑO DE USER EXPERIENCE AL CICLO DEVOPS CI/CD**, presentado por: **André Mendoza Torres** después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Firmado electrónicamente por: José Francisco
Gómez Rivera
Motivo: Orden de impresión
Fecha: 14/07/2023 19:20:42
Lugar: Facultad de Ingeniería, USAC.

Ing. José Francisco Gómez Rivera
Decano a.i.



Guatemala, julio de 2023

Para verificar validez de documento ingrese a <https://www.ingenieria.usac.edu.gt/firma-electronica/consultar-documento>

Tipo de documento: Correlativo para orden de impresión Año: 2023 Correlativo: 31 CUI: 2903883170101

Escuelas: Ingeniería Civil, Ingeniería Mecánica Industrial, Ingeniería Química, Ingeniería Mecánica Eléctrica, - Escuela de Ciencias, Regional de Ingeniería Sanitaria y Recursos Hidráulicos (ERIS). Postgrado Maestría en Sistemas Mención Ingeniería Vial. Carreras: Ingeniería Mecánica, Ingeniería Electrónica, Ingeniería en Ciencias y Sistemas. Licenciatura en Matemática. Licenciatura en Física. Centro de Estudios Superiores de Energía y Minas (CESEM). Guatemala, Ciudad

ACTO QUE DEDICO A:

Dios

Por las bendiciones recibidas.

Mis padres

German Estuardo Mendoza Reyes y Annabella Torres Haeussler de Mendoza, por su apoyo a lo largo de mi vida y por brindarme todo lo necesario en mi formación profesional.

Mi hermana

Michelle Mendoza Torres, por animarme a seguir adelante a pesar de los obstáculos que pudieran existir.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala Por ser la casa de estudios que permitió mi acceso a la educación superior.

Mis amigos de la universidad Por las experiencias vividas juntos.

Mi asesor Ing. José Manuel Ruiz Juárez por brindarme su apoyo en la elaboración de esta tesis.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES.....	VII
GLOSARIO	IX
RESUMEN.....	XI
OBJETIVOS.....	XIII
INTRODUCCIÓN	XV
1. DESARROLLO DE SOFTWARE.....	1
1.1. Definición.....	1
1.2. Metodología.....	2
1.2.1. <i>Waterfall</i> (Cascada).....	2
1.2.1.1. Recopilación de requisitos.....	3
1.2.1.2. Análisis	4
1.2.1.3. Comprender los requisitos técnicos.....	4
1.2.1.4. Desarrollo	4
1.2.1.5. Prueba o validación	4
1.2.1.6. Lanzamiento del producto.....	5
1.3. <i>Framework</i> (marco de trabajo)	6
1.3.1. Scrum	6
1.3.2. XP (<i>Extreme Programming</i> , Programación Extrema).....	9
1.3.3. Kanban	13
1.4. Ágil	15
1.4.1. ¿Qué problema resuelve Ágil?	17
1.4.2. Manifiesto Ágil	17
1.4.2.1. Valores	17

	1.4.2.2.	Principios.....	18
1.4.3.		Prácticas.....	19
	1.4.3.1.	Sprints	19
	1.4.3.2.	Daily standup.....	20
	1.4.3.3.	<i>User stories</i> (historias de usuario).....	21
	1.4.3.4.	<i>Pair programming</i> (programación en pareja o programación en pares)	22
	1.4.3.5.	<i>Test-driven development</i> (TDD, desarrollo guiado por pruebas).....	23
1.4.4.		Beneficios.....	24
1.5.		Niveles de automatización	25
	1.5.1.	Desarrollo Ágil.....	26
	1.5.2.	<i>Continuous Integration</i> (Integración continua).....	26
	1.5.3.	<i>Continuous Testing</i> (Pruebas continuas)	27
	1.5.4.	<i>Continuous Delivery</i> (Entrega continua).....	27
	1.5.5.	<i>Continuous Deployment</i> (Despliegue continuo, Implementación continua)	27
	1.5.6.	<i>Continuous Monitoring</i> (Monitoreo continuo).....	27
	1.5.7.	Ciclo DevOps CI/CD.....	28
1.6.		CI/CD	28
2.		DEVOPS.....	29
	2.1.	Definición	29
	2.2.	Desarrollo de software con métodos tradicionales.....	30
	2.2.1.	Conflicto	32
	2.3.	Muro de confusión.....	32
	2.4.	Ciclo DevOps CI/CD.....	33
	2.4.1.	Beneficios.....	35
	2.4.2.	Pasos	36

	2.4.2.1.	<i>Plan</i> (planificación)	36
	2.4.2.2.	<i>Code</i> (codificación)	36
	2.4.2.3.	<i>Build</i> (construcción)	37
	2.4.2.4.	<i>Test</i> (pruebas)	37
	2.4.2.5.	<i>Release</i> (lanzamiento o liberación).....	37
	2.4.2.6.	<i>Deploy</i> (despliegue).....	37
	2.4.2.7.	<i>Operate</i> (operación).....	38
	2.4.2.8.	<i>Monitor</i> (monitoreo)	38
2.5.		Efectividad	38
3.		<i>USER EXPERIENCE</i>	41
3.1.		Definición.....	41
3.2.		Diseño de UX	41
	3.2.1.	<i>UX Umbrella</i> (Paraguas UX).....	41
	3.2.1.1.	<i>Visual design</i> (diseño visual)	42
	3.2.1.2.	<i>Information architecture</i> (arquitectura de la información)	42
	3.2.1.3.	<i>Interaction design</i> (diseño de interacción)	43
	3.2.1.3.1.	Palabras	43
	3.2.1.3.2.	Representaciones visuales	43
	3.2.1.3.3.	Objetos o espacios físicos	44
	3.2.1.3.4.	Tiempo	44
	3.2.1.3.5.	Comportamiento	44
	3.2.1.4.	<i>Usability</i> (usabilidad).....	44
	3.2.1.4.1.	<i>Learnability</i> (capacidad de aprendizaje).....	45

	3.2.1.4.2.	<i>Efficiency</i> (eficiencia)	45
	3.2.1.4.3.	<i>Memorability</i> (hecho memorable)	45
	3.2.1.4.4.	<i>Errors</i> (errores).....	45
	3.2.1.4.5.	<i>Satisfaction</i> (satisfacción)	45
	3.2.1.5.	<i>User research</i> (investigación de usuarios)	46
	3.2.1.6.	<i>Content strategy</i> (estrategia de contenido).....	47
3.3.	Proceso de diseño de UX.....		48
3.3.1.	<i>Design Thinking</i> (Pensamiento de Diseño)		49
	3.3.1.1.	<i>Empathize</i> (empatizar)	50
	3.3.1.2.	<i>Define</i> (definir).....	50
	3.3.1.3.	<i>Ideate</i> (idear)	51
	3.3.1.4.	<i>Prototype</i> (prototipo).....	51
	3.3.1.5.	<i>Test</i> (prueba).....	51
3.3.2.	Lean UX		52
	3.3.2.1.	<i>Think</i> (pensar)	53
	3.3.2.2.	<i>Make</i> (hacer)	54
	3.3.2.3.	<i>Check</i> (comprobar).....	54
3.3.3.	<i>Design Sprint 2.0</i> (Sprint de Diseño 2.0)		54
	3.3.3.1.	Lunes.....	55
	3.3.3.1.1.	<i>Define the challenge</i> (definir el desafío)	56
	3.3.3.1.2.	<i>Produce solutions</i> (producir soluciones).....	57
	3.3.3.2.	Martes	59

	3.3.3.2.1.	<i>Vote on solutions</i> (votar soluciones)	60
	3.3.3.2.2.	<i>The storyboard</i> (guion gráfico)	62
	3.3.3.3.	Miércoles	62
	3.3.3.4.	Jueves	63
3.4.		Perfiles de proceso de diseño de UX	64
	3.4.1.	Perfil <i>software houses</i> (casas de software)	65
	3.4.2.	Perfil software propio	66
4.		DISEÑO DE UX Y DESARROLLO DE SOFTWARE.....	69
	4.1.	Necesidad del diseño de UX en el desarrollo de software.....	69
	4.2.	Colaboración entre diseñadores y desarrolladores	71
	4.2.1.	Consejos de colaboración para diseñadores	72
		4.2.1.1. Incluir a los desarrolladores de principio a fin.....	72
		4.2.1.2. Documentar los diseños	73
		4.2.1.3. Crear prototipos	73
		4.2.1.4. Ser muy organizado.....	74
		4.2.1.5. Adaptar sus procesos	74
	4.3.	Colaboración entre diseñadores de UX y <i>product managers</i> ..	75
		4.3.1. <i>Product manager</i> (gerente de producto).....	76
		4.3.2. Diseñador de UX	78
		4.3.3. <i>Product manager</i> y diseñador de UX.....	78
	4.4.	Relación entre DevOps y diseño de UX	80
		4.4.1. Habilidades necesarias para ser un ingeniero de DevOps y para ser un diseñador de UX.....	80
		4.4.2. Principios de DevOps y diseño de UX.....	81

- 5. INTEGRACIÓN DEL PROCESO DE DISEÑO DE *USER EXPERIENCE*
AL CICLO DEVOPS CI/CD83
 - 5.1. UXDevOps83
 - 5.1.1. Equipos en UXDevOps.....85
 - 5.1.2. Proceso en UXDevOps87

- CONCLUSIONES.....91
- RECOMENDACIONES93
- REFERENCIAS95
- APÉNDICES.....99

ÍNDICE DE ILUSTRACIONES

FIGURAS

1.	<i>Waterfall</i>	3
2.	Metodología <i>Waterfall</i>	5
3.	Scrum.....	6
4.	Fases Scrum.....	9
5.	<i>Extreme Programming</i>	10
6.	Fases <i>Extreme Programming</i>	13
7.	Kanban.....	14
8.	Tablero Kanban.....	15
9.	<i>Sprints</i>	20
10.	<i>Daily standup</i>	21
11.	<i>User stories</i>	22
12.	<i>Pair programming</i>	23
13.	<i>Test-driven development</i>	24
14.	Niveles de automatización.....	26
15.	Desarrollo con métodos tradicionales	30
16.	Control de calidad con métodos tradicionales	31
17.	Operaciones con métodos tradicionales.....	32
18.	Muro de confusión.....	33
19.	Ciclo DevOps CI/CD.....	35
20.	Organizaciones de mayor rendimiento	40
21.	<i>UX Umbrella</i>	42
22.	Piezas de diseño de Experiencia de Usuario	48
23.	<i>Design Thinking</i>	50

24.	<i>Lean UX</i>	53
25.	<i>Design Sprint 2.0</i>	55
26.	Lunes <i>Design Sprint 2.0</i>	56
27.	Martes <i>Design Sprint 2.0</i>	60
28.	Perfil software propio.....	67
29.	Desarrollo de software sin proceso de diseño de UX	70
30.	Desarrollo de software con proceso de diseño de UX	71
31.	UXDevOps	83
32.	Partes UXDevOps.....	85
33.	Equipos en UXDevOps	86
34.	Ajustes Proceso UXDevOps	87
35.	Riesgos Proceso UXDevOps	88
36.	Proceso UXDevOps.....	89

TABLAS

I.	Métricas de rendimiento clave	39
II.	Jueves formato observar y tomar notas <i>Design Sprint 2.0</i>	64
III.	Habilidades necesarias para ser un ingeniero de Devops y para ser un diseñador de UX	81

GLOSARIO

Ambiente de producción	Entorno donde se ejecuta la última versión funcional del software y se pone a disposición de los usuarios finales.
Código confirmado	Código enviado a un repositorio compartido.
Confirmar	Enviar código a un repositorio compartido.
Desplegar	Enviar código al servidor de producción y ejecutarlo.
Lanzar	Enviar código que pasa las pruebas automatizadas a un repositorio compartido en un servidor.
<i>Return on investment</i>	Retorno de la inversión. Métrica utilizada para evaluar el beneficio o la pérdida en una inversión.
<i>Software as a Service</i>	Software como servicio. Software alojado por un proveedor externo que es ofrecido como servicio a través de Internet a los clientes. Generalmente se suele pagar una suscripción para utilizar el servicio.
<i>Startup</i>	Empresa emergente con gran potencial de crecimiento, centrada en un crecimiento exponencial.

RESUMEN

La manera en la que se construye el software ha cambiado varias veces con el tiempo, adaptando e integrando paradigmas que no siempre surgen en el desarrollo de software. Dentro de estos paradigmas surge la mentalidad DevOps que se vuelve bastante popular ya que resuelve el problema de comunicación entre el área de desarrollo y el área de operaciones, con el tiempo se aplica a la integración y despliegue continuo, dando como resultado el Ciclo DevOps CI/CD, un ciclo bastante óptimo para desarrollar software.

El desarrollo de software está estrechamente relacionado con el diseño de UX. El diseño de UX se encarga de diseñar la experiencia que un usuario tiene al interactuar con un producto, en este caso un producto digital. Los métodos, *frameworks* y prácticas utilizadas para el diseño de UX muchas veces tienen el mismo origen o comparten los mismos principios que otros métodos, *frameworks* o prácticas en el desarrollo de software.

El proceso de diseño de UX y el Ciclo DevOps CI/CD, ambos tienen como uno de sus objetivos principales entregar un buen software. El integrar el proceso de diseño de UX al Ciclo DevOps CI/CD da como resultado un ciclo para el desarrollo de software en donde se toman en cuenta ambas perspectivas.

OBJETIVOS

General

Proponer un modelo en el que el proceso de diseño de *User Experience* se integre al Ciclo DevOps CI/CD, mediante el uso de prácticas existentes, para facilitar el proceso de desarrollo de software.

Específicos

1. Definir el concepto de DevOps de acuerdo con la comunidad de desarrolladores y diseñadores.
2. Definir el concepto de *User Experience* en un contexto digital de acuerdo con la comunidad de desarrolladores y diseñadores.
3. Establecer la relación entre DevOps y *User Experience* en el desarrollo de software.
4. Proponer un Ciclo DevOps CI/CD el cual incluya el proceso de diseño de *User Experience* enlazando prácticas existentes.

INTRODUCCIÓN

Actualmente, para la gran mayoría de personas la utilización de software consciente o inconscientemente es imprescindible en el día a día, ya sea que este se utilice como herramienta o como entretenimiento. Lo que hace que exista una gran demanda de software y que los clientes y los usuarios quieran nuevo software, nuevas funcionalidades y actualizaciones cada vez más rápido.

No se podría solventar esta demanda si no existieran métodos, *frameworks* y prácticas para el desarrollo de software que permitan mantener un orden y una dirección para llegar a los resultados deseados, dentro de estos uno de los más populares es el Ciclo DevOps CI/CD. Y para que esto se cumpla de la manera más óptima posible, todo desarrollo de software debe de incluir un proceso diseño de UX en el cual también existen métodos, *frameworks* y prácticas.

Este trabajo en sus inicios expone los métodos, *frameworks* y prácticas utilizadas en el desarrollo de software y en el diseño de UX y como estos se utilizan.

Ya que se tiene una comprensión de los temas por tratar, se establece la relación y necesidad que existe entre el desarrollo de software y el diseño de UX. Además de establecer ciertas prácticas para que se tenga una buena colaboración entre las distintas partes.

Para terminar, se plantea un ciclo en el que se integra el proceso de diseño de UX al Ciclo DevOps CI/CD. Y se describe la manera correcta en la que este ciclo debe de ejecutarse.

1. DESARROLLO DE SOFTWARE

1.1. Definición

El desarrollo de software es fundamentalmente pasar de una idea abstracta a un sistema concreto que realmente implementa esa idea, e idealmente, lo hace de la manera correcta.

El desarrollo de software es un proceso bastante complejo a menos que se esté hablando de un sistema trivial. Por lo tanto, tener un proceso de desarrollo de software es de fundamental importancia.

Un proceso de desarrollo software es una forma de dividir una tarea en pasos más pequeños, que se pueden manejar y abordar individualmente. Desglosar la complejidad de una manera sistemática. Por lo tanto, los procesos de software también son una forma formal o semiformal de discutir o describir cómo se debe desarrollar el software.

Los procesos de desarrollo de software se utilizan con el fin de reducir el tiempo y el costo de desarrollar productos de software, con estos procesos se diseñan productos en torno a las necesidades de los clientes de una manera eficiente.

No existe un único proceso de desarrollo de software que se pueda utilizar, sino múltiples procesos posibles, dependiendo del contexto. Es posible utilizar una combinación de varios procesos.

1.2. Metodología

Una metodología es un enfoque documentado para realizar actividades de una manera coherente, consistente, responsable y repetible.

Una metodología dice qué hacer y cómo hacerlo, es un conjunto de reglas, reglas que se deben seguir para lograr un objetivo final, estas reglas son rígidas y las personas que siguen una metodología no pueden cambiar esas reglas.

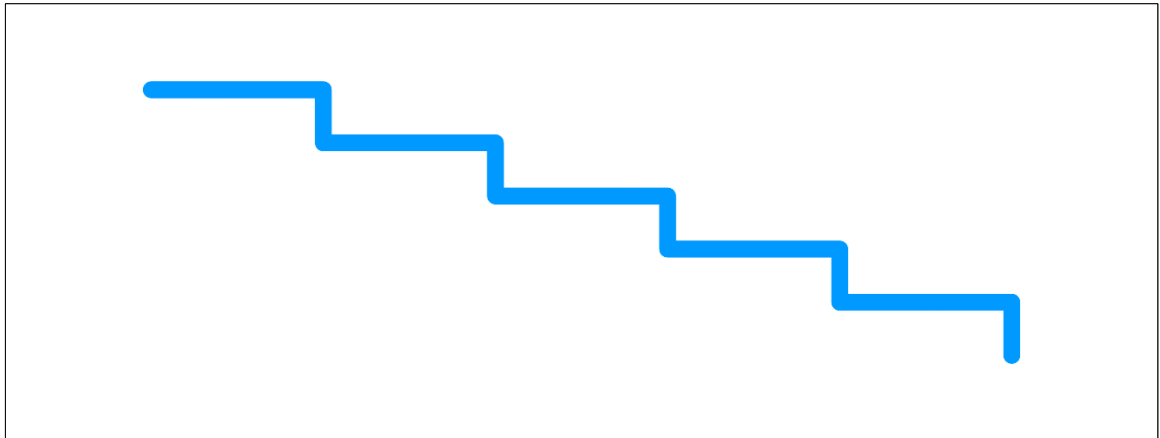
1.2.1. *Waterfall* (Cascada)

Waterfall es una metodología que destaca el conjunto de prácticas que se requieren y se espera que sigan los gerentes de proyecto.

En su conjunto promueve un ciclo de vida de desarrollo de software (SDLC, *Systems Development Life Cycle*) muy poco flexible.

El modelo en cascada enfatiza que los proyectos deben seguir una progresión lógica de pasos a lo largo del ciclo de vida de desarrollo de software y, tal como su nombre lo indica, cada fase del proyecto cae en cascada hacia la siguiente, que fluye progresivamente hacia abajo como una cascada.

Figura 1. **Waterfall**



Fuente: elaboración propia, realizado con Microsoft Visio.

Los proyectos se pueden dividir en varias fases distintas, este es un conjunto ordenado de fases y cada fase debe completarse una por una, la fase dos no se puede iniciar hasta que se haya completado la fase anterior y así sucesivamente.

Consta de varias fases:

1.2.1.1. Recopilación de requisitos

Los requisitos potenciales del producto se recopilan metódicamente y se anotan en un documento de especificaciones, con este documento de requisitos los gerentes de proyecto (*project managers*) planifican todas las demás fases sin más correspondencia con el cliente hasta que el producto esté completo. Se supone que toda la recopilación de requisitos ocurre en esta fase, y en esta fase los requisitos son recopilados por el analista de negocios (*business analyst*) y se analizan.

1.2.1.2. Análisis

Con una compilación de ideas que definen lo que debe hacer la aplicación, ahora se transforman en un plan accionable. Durante esta etapa se deben generar adecuadamente los modelos en la lógica de negocio que se utilizarán en la aplicación

1.2.1.3. Comprender los requisitos técnicos

Se convierten las necesidades del negocio en requisitos técnicos. En esta fase se deben crear especificaciones de diseño para describir cómo exactamente se implementarán las necesidades del negocio cubiertas en el documento de requisitos desde un punto de vista técnico. Este proceso de diseño cubre los requisitos de diseño técnico, como el lenguaje de programación, las capas de datos, los servicios y más.

1.2.1.4. Desarrollo

Esta es la fase de implementación cuando finalmente se escribe el código fuente real. Se implementan todos los modelos, requisitos de negocio e integraciones que se especificaron en las etapas anteriores.

1.2.1.5. Prueba o validación

En esta fase de prueba los *qa testers* y *beta testers* localizan e informan sistemáticamente los problemas dentro de la aplicación. Desde esta etapa el proyecto a menudo vuelve a la etapa anterior, a veces incluso vuelve al diseño en la fase de entender donde se reescribe el código para garantizar que se

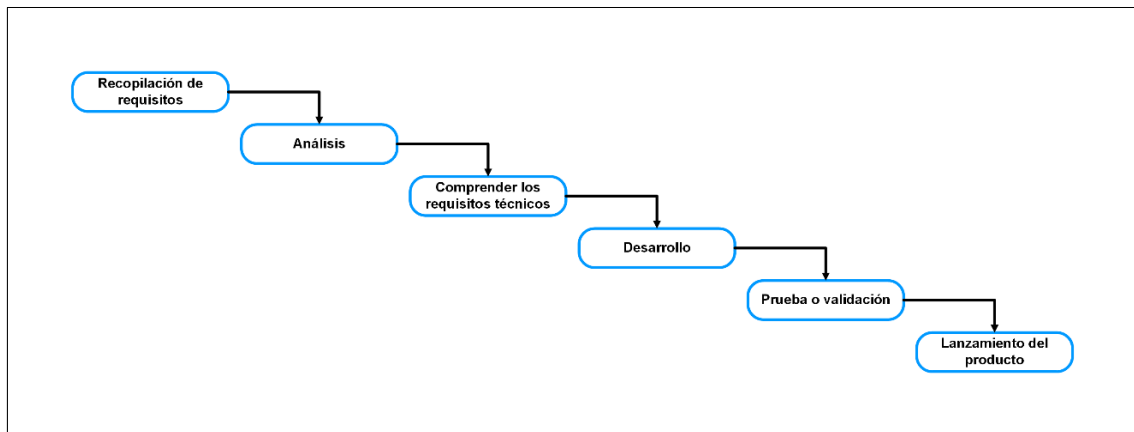
corrijan los errores. En este caso, el proyecto debe pasar por pruebas nuevamente hasta que los *testers* lo aprueben.

1.2.1.6. Lanzamiento del producto

Una vez que se ha probado un producto, luego pasa a la sexta y última etapa. En esta etapa, la aplicación está lista para su implementación en un entorno en vivo donde los usuarios pueden acceder a ella y responder con más *feedback*. Es clave que los clientes revisen el producto para asegurarse de que cumple con los requisitos establecidos al principio del proyecto.

Después del lanzamiento del producto, es importante mantener el soporte posterior en el producto para mantenerlo funcional y actualizado.

Figura 2. Metodología *Waterfall*



Fuente: elaboración propia, realizado con Microsoft Visio.

1.3. **Framework (marco de trabajo)**

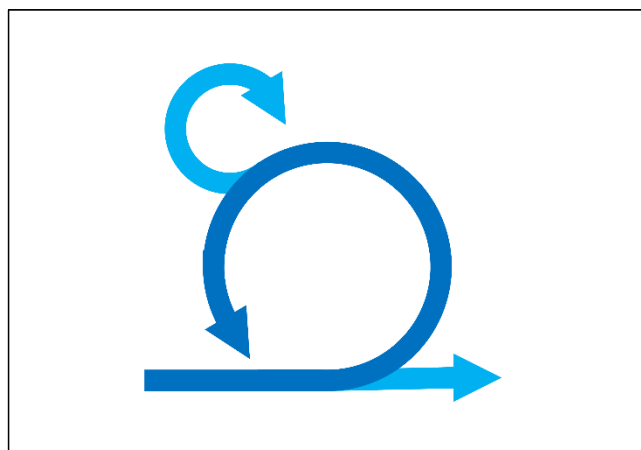
Un *framework* es un conjunto de pautas poco acopladas. Un *framework* dice qué hacer, pero deja el cómo hacerlo en quien lo usa.

Existen muchos *frameworks* dentro de los cuales se puede decidir cuál es el que mejor se adapta a una situación en concreto, dentro de los *frameworks* más populares que existen están:

1.3.1. **Scrum**

Scrum es un *framework* ágil orientado a objetivos, utilizado en el desarrollo que busca aprovechar las prácticas iterativas e incrementales para ayudar a administrar productos altamente complejos. Scrum es una de las principales ramas o subconjuntos de Ágil

Figura 3. **Scrum**



Fuente: elaboración propia, realizado con Microsoft Visio.

Los roles destacados en Scrum son:

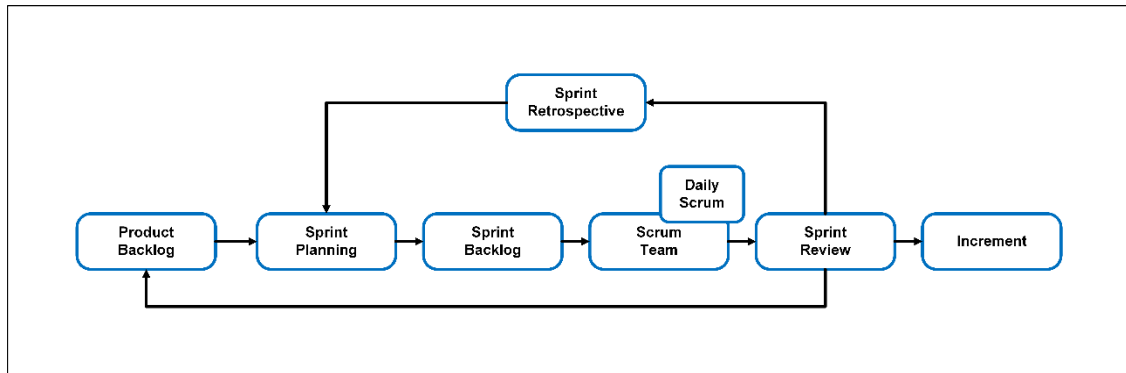
- *Product Owner* (propietario del producto). Es el responsable de establecer la comunicación entre el *development team* (equipo de desarrollo) y los clientes. El *Product Owner* se asegura de que las expectativas de un producto completo se comuniquen y acuerden claramente.
- *Scrum Master*. Son los expertos en Scrum dentro de sus equipos. Capacitan y entrenan a los equipos y *product owners*. También refinan las prácticas y mejoran los flujos de trabajo y entrega.
- *Development team* (equipo de desarrollo). Los miembros del equipo son responsables del desarrollo de las versiones del producto final. Estos profesionales están bien equipados con el conocimiento de Scrum y prácticas ágiles, a menudo, son profesionales certificados con certificaciones de Scrum reconocidas en la industria.

Funcionamiento:

- Se tiene el *product backlog* que es una lista de tareas que deben completarse para que se logren los objetivos de los *stakeholders*.
- En el *sprint planning* se decide qué tareas del *product backlog* se deben realizar y entregar en un período de 2 a 4 semanas llamado *sprint*.
- Las tareas discutidas en el *sprint planning* se agregan al *sprint backlog*, este es un conjunto de tareas que se trabajaran en el *sprint* en curso.

- El equipo de Scrum que generalmente tiene entre cinco y nueve miembros trabajará en las tareas del *sprint backlog*.
- Se tiene la *daily scrum*, que es una reunión interna diaria de 15 minutos, donde cada miembro del equipo habla sobre que hizo ayer, que hará hoy y si existe algún obstáculo o impedimento que este bloqueando su progreso.
- Al finalizar el *sprint* se tienen el *sprint review*. El *sprint review* es una reunión durante la cual el equipo muestra lo que logró durante el *sprint*. Durante este tiempo se hacen preguntas, observaciones, comentarios y sugerencias. El *product owner* también recibe comentarios de los *stakeholders* para los próximos *sprints*.
- Luego del *sprint review* se tiene un *sprint retrospective*, donde se identifican errores pasados, problemas potenciales y nuevas formas de manejarlos, los nuevos datos se incorporan al nuevo *sprint plan*.
- El paso final es incrementar, donde se proporciona un resultado viable y utilizable a los *stakeholders*.

Figura 4. Fases Scrum



Fuente: elaboración propia, realizado con Microsoft Visio.

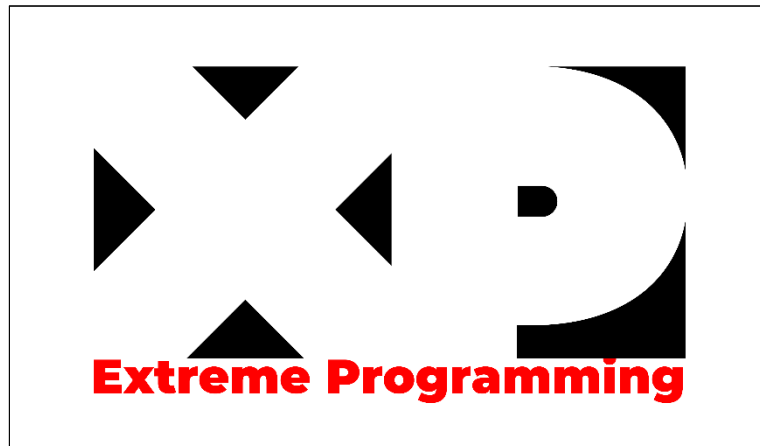
1.3.2. XP (*Extreme Programming, Programación Extrema*)

La Programación Extrema es un *framework* ligero y humanista de desarrollo de software. Tiene como objetivo permitir que los equipos pequeños y medianos produzcan software de alta calidad y se adapten a los requisitos cambiantes o en evolución. Se centra más en el análisis, planificación y codificación en lugar de reuniones, horarios complicados y documentación. Las tareas se realizan de persona a persona por lo que es un *framework* más humanista.

La razón del nombre Programación Extrema es porque lleva las prácticas de desarrollo a un nivel extremo.

XP es ideal en contextos donde se tienen entregas frecuentes, los requisitos cambian constantemente y el cliente pide cambios con mucha frecuencia.

Figura 5. ***Extreme Programming***



Fuente: elaboración propia, realizado con Microsoft Visio.

Lo valores de XP son: comunicación, simplicidad, retroalimentación, coraje y respeto. Los principios básicos de XP son: retroalimentación rápida, asumir simplicidad, cambios incrementales, abrazar los cambios y trabajo de alta calidad.

Las prácticas de XP incluyen:

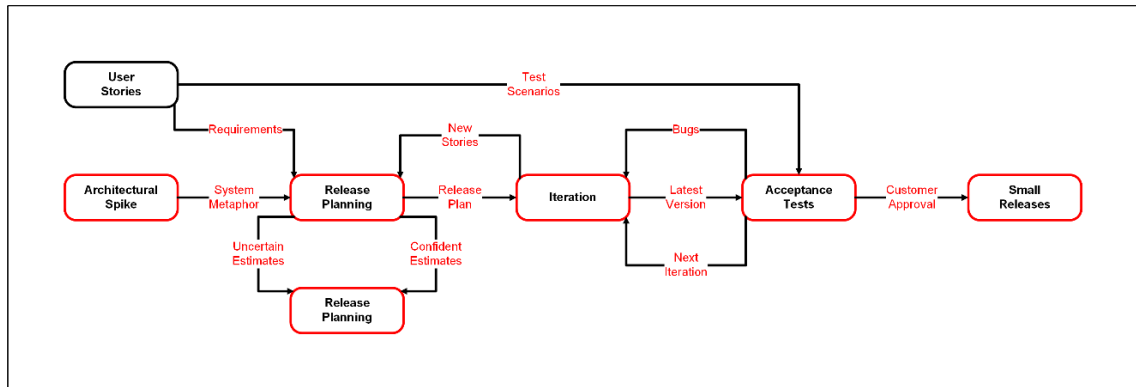
- Juego de planificación. Se realiza en interacciones semanales, al comienzo de la semana los desarrolladores y los clientes se reúnen para priorizar las funcionalidades. Ya deben de haber sido creadas las *user stories*.
- Pequeñas entregas. Son pequeñas piezas funcionales del producto que se entregan al cliente antes de que el producto esté terminado. Ayuda en el proceso de aceptación por parte del cliente ya que puede probar parte del sistema que está comprando.

- Metáfora. Busca facilitar la comunicación con el cliente entendiendo cuál es la realidad de esta, es necesario traducir las palabras del cliente al significado que espera dentro del proyecto.
- Diseño simple. Es básicamente realizar lo que el cliente pide y no agregar funcionalidades extras.
- Pruebas de clientes. Una prueba de cliente es una prueba construida por el cliente y un conjunto de analistas y *testers* para aceptar un determinado requisito del sistema.
- Ritmo de trabajo sostenible. Es trabajar con calidad buscando tener un ritmo de trabajo saludable. Establecer cierta cantidad de horas al día que permitan trabajar con productividad en el proyecto.
- Propiedad colectiva. El código fuente no tiene dueño y nadie necesita solicitar permiso para poder modificar el mismo, el objetivo con esto es hacer que el equipo conozca todas las partes del sistema.
- Programación en pares. Dos desarrolladores codificando juntos en un solo ordenador. De esta manera el código siempre se revisa por dos personas, mejorando la calidad del código fuente generado, reduciendo la posibilidad de defectos. La programación en pares es una de las prácticas principales de XP.
- Estandarización de código. El equipo de desarrollo necesita establecer reglas para programar y todos deben seguir estas reglas, de esta manera parecerá que todo el código fuente fue escrito por la misma persona, incluso cuando el equipo tenga muchos miembros. La estandarización de

código es muy importante, dado que si cada miembro programa de una manera distinta, el código se vuelve muy confuso y en el futuro se pueden tener problemas para revisarlo y entenderlo.

- Desarrollo Orientado a Pruebas (TDD, *Test Driven Development*). Se crean las pruebas unitarias y luego se crea un código que pase las pruebas unitarias. Las pruebas unitarias son esenciales para que la calidad del proyecto se mantenga.
- Refactorización. Proceso que permite la mejora continua del código, la mínima entrada de errores y la compatibilidad con el código existente. La refactorización mejora la claridad de la lectura del código, divide el código en módulos más cohesivos y reutilizables y evita la duplicación de código fuente.
- Integración Continua (*Continuous Integration*). Cada vez que se produce una nueva funcionalidad debe de integrarse lo más pronto posible a la versión actual del sistema y no esperar a integrar cuando se complete la iteración, esto solo aumenta la posibilidad de conflictos y la posibilidad de errores en el código fuente.

Figura 6. Fases *Extreme Programming*



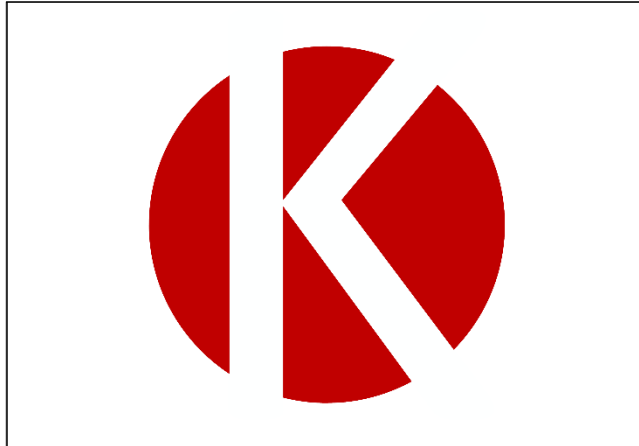
Fuente: elaboración propia, realizado con Microsoft Visio.

1.3.3. Kanban

Kanban es un *framework* ágil con un sistema visual mediante el cual el trabajo se puede administrar con facilidad a medida que avanza.

Kanban se enfoca en reducir la multitarea, disminuir el desperdicio general producido, poner las necesidades del cliente primero y básicamente el retorno de la inversión (ROI, *return on investment*).

Figura 7. **Kanban**



Fuente: elaboración propia, realizado con Microsoft Visio.

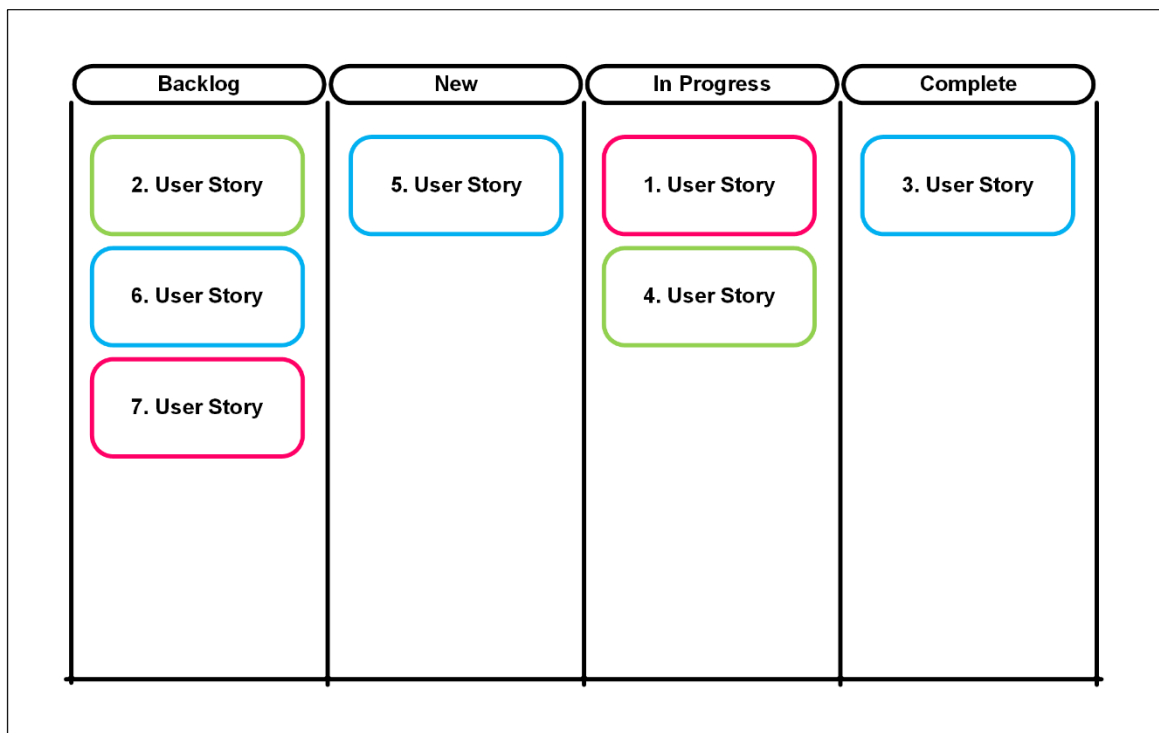
Kanban usa algo conocido como tablero Kanban, con esto se pueden identificar fácilmente los cuellos de botella y luego solucionarlos de manera rentable a velocidades óptimas.

Los sistemas Kanban comienzan con un tablero y tarjetas visuales que representan elementos del *product backlog*. En el tablero, se colocan las tarjetas en columnas que representen el paso actual en el flujo de trabajo, que va desde "Nuevo" hasta "Completo". Los pasos intermedios dependen de quien lo utiliza, se debe de mantener simple y eficiente.

Para ayudar a garantizar que los elementos se completen a un ritmo constante, Kanban impone límites en la cantidad de elementos que pueden vivir en cualquier paso del flujo de trabajo en un momento dado. Estos se denominan *Work in Progress* (trabajo en curso) o límites WIP. Deben configurarse de manera que el trabajo se desarrolle de la manera más fluida y consistente posible.

Limitar la cantidad de trabajo que está en progreso significa que se deben terminar algunas tareas antes de poder comenzar con elementos adicionales. Si existiera algún problema en el flujo de trabajo estos límites lo evidenciaran rápidamente al crear cuellos de botella fácilmente visibles.

Figura 8. **Tablero Kanban**



Fuente: elaboración propia, realizado con Microsoft Visio.

1.4. **Ágil**

Ágil es una mentalidad que permite navegar alrededor de restricciones y complejidades desconocidas de manera eficiente.

Ágil se basa en la creencia de que las incertidumbres se pueden manejar con éxito y precisión al detectar eventos en tiempo real y responder en consecuencia antes del último momento responsable. Esto libera de toda regla, límite e incertidumbre al saber que no importa lo que venga siempre hay una manera para navegar a su alrededor.

Ágil no es prescriptivo y no está sujeto a ninguna regla, brindando la libertad para elegir cualquier método o procedimiento existente o incluso crear propios procedimientos o métodos que se adapten a una situación y circunstancias.

La mentalidad Ágil se puede aplicar fácilmente a cualquier función empresarial, ya sea recursos humanos, *marketing*, análisis de negocios o desarrollo de software, incluso en el ámbito personal. La mentalidad Ágil proporciona orientación sobre cómo percibir el cambio y responder a él, y sobre cómo lidiar con la incertidumbre.

Ágil no es una metodología y no es un *framework*.

Ágil está formado por un conjunto de valores y principios descritos en el Manifiesto Ágil.

Para trabajar de una manera Ágil, cualquier decisión que se tome tiene que estar basada en los valores y principios establecidos por Ágil; de lo contrario, no se está trabajando de una manera Ágil y, por lo tanto, no se obtendrán los beneficios de Ágil.

1.4.1. ¿Qué problema resuelve Ágil?

Usando la metodología *Waterfall* se tiene muy poca flexibilidad para desarrollar un software, en la mayoría de los casos eso no es factible, ya que la demanda de software es muy alta y se necesita poder cambiar de acuerdo con las necesidades del mercado, las metodologías tradicionales de desarrollo de software simplemente no están construidas para esto. Para poder hacer frente a esta problemática es que surge Ágil.

Ágil da la capacidad de cambiar y adaptarse al mercado para satisfacer las necesidades de este y hacerlo en un plazo muy corto, lo que hace que Ágil sea muy adecuado para desarrollar software.

1.4.2. Manifiesto Ágil

El Manifiesto Ágil es un breve documento basado en los cuatro valores y doce principios del Desarrollo Ágil de Software.

1.4.2.1. Valores

Beck, (s.f.) Los cuatro valores del Manifiesto Ágil incluyen:

- 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 se valoren los elementos de la derecha, se valoran más los de la izquierda. (párr. 2)

1.4.2.2. Principios

Beck, (s.f.) plantea que los doce principios del Manifiesto Ágil incluyen:

- Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
- Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
- 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.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software que funciona es la medida principal de progreso.
- Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.

- La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia. (párr. 1)

1.4.3. Prácticas

Existen muchas prácticas Ágiles las cuales se deben de elegir en función de las necesidades de cada caso en particular.

Algunas de las prácticas Ágiles son:

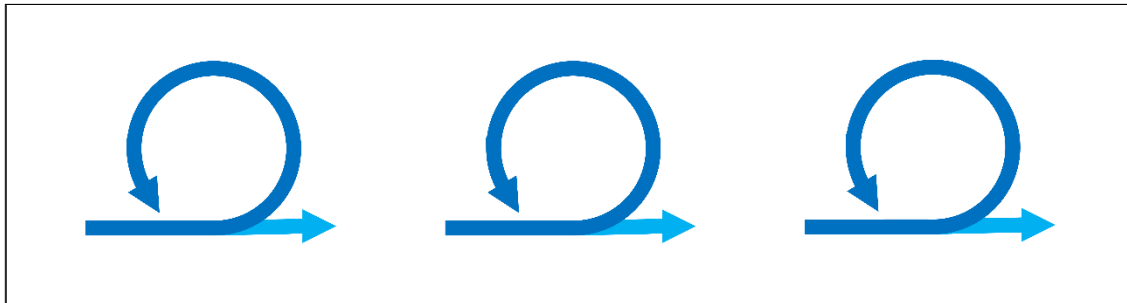
1.4.3.1. Sprints

Un *sprint* es un período de tiempo definido de no más de 1 mes que un equipo tiene para completar una cantidad fija de trabajo.

Aunque no hay un límite de tiempo inferior prescrito, se considera que la duración mínima del *sprint* es de 1 semana.

El tiempo definido suele establecerse en número de semanas, siendo 1 semana el tiempo mínimo y 4 semanas el tiempo máximo.

Figura 9. **Sprints**



Fuente: elaboración propia, realizado con Microsoft Visio.

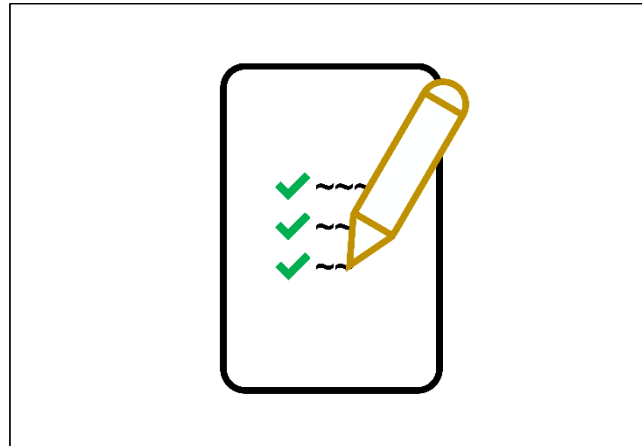
1.4.3.2. **Daily standup**

Son reuniones diarias de 15 minutos entre los miembros del equipo. Mantienen al equipo al tanto y enfocado en el objetivo, además de que proporcionan una forma temprana de detectar cualquier riesgo.

Los miembros del equipo se turnan para responder alguna variación de tres preguntas rápidas y simples: ¿qué hiciste ayer? ¿qué harás hoy? y ¿qué obstáculos estás experimentando actualmente?

Estas reuniones generalmente se realizan al comienzo del día, aunque pueden hacerse a la hora que más convenga, preferiblemente a la misma hora y en el mismo lugar.

Figura 10. ***Daily standup***



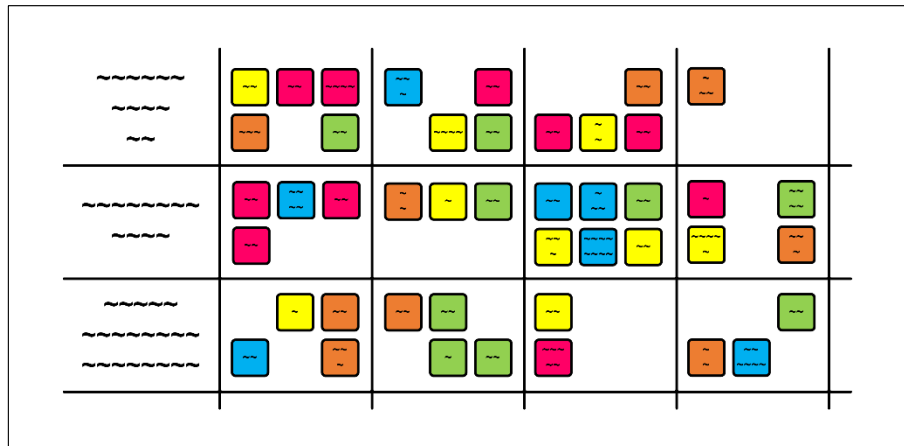
Fuente: elaboración propia, realizado con Microsoft Visio.

1.4.3.3. *User stories* (historias de usuario)

Una historia de usuario es una breve descripción de una funcionalidad que tiene valor para el cliente, no es más que la historia detrás del ¿por qué? un usuario necesita algo.

La historia de usuario proporciona valor comercial y es algo que un usuario final entiende.

Figura 11. **User stories**



Fuente: elaboración propia, realizado con Microsoft Visio.

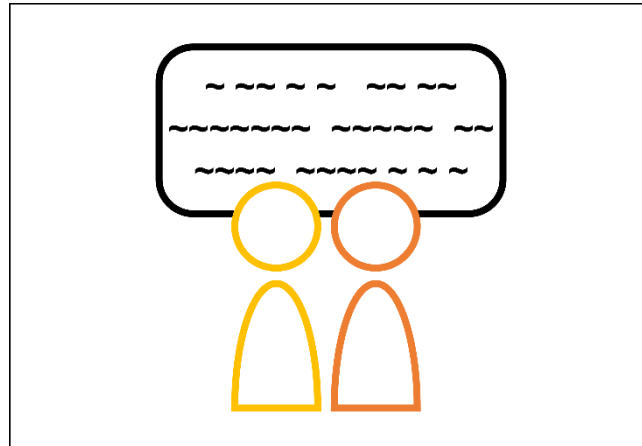
1.4.3.4. **Pair programming (programación en pareja o programación en pares)**

Es una práctica en la que dos programadores trabajan juntos en la misma computadora para escribir código. Hay dos roles: un conductor (*driver*) y un navegador (*navigator*). El conductor es quien usa la computadora, quien en realidad escribe el código y el navegador es quien realiza un seguimiento del panorama general y ayuda al conductor a decidir qué código escribir a continuación.

Es importante cambiar de rol a menudo para poder practicar ambos roles y aprender unos de otros.

El hecho de que dos pares de ojos miren el código al mismo tiempo, reduce significativamente la posibilidad de que algo salga mal con él.

Figura 12. ***Pair programming***



Fuente: elaboración propia, realizado con Microsoft Visio.

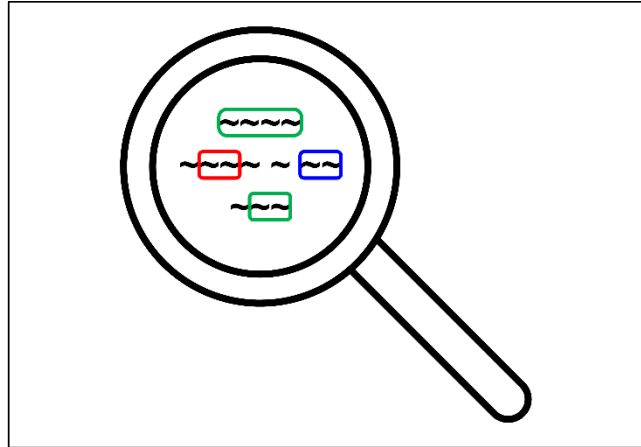
1.4.3.5. *Test-driven development* (TDD, desarrollo guiado por pruebas)

Es una práctica en la que primero se escribe la prueba fallida y luego se escribe el código para resolver el problema que hace que la prueba pase.

TDD se da en un ciclo de tres pasos:

- Rojo, se comienza escribiendo una prueba fallida.
- Verde, se escribe el código justo para que la prueba pase y nada más. En este momento no importa qué tan mala o buena sea la solución.
- Refactorizar (azul), finalmente se comienza a refactorizar la solución para mejorarla en muchos aspectos, limpiando el código y haciendo mejores diseños mientras se mantiene la prueba pasando.

Figura 13. ***Test-driven development***



Fuente: elaboración propia, realizado con Microsoft Visio.

1.4.4. Beneficios

- **Transparencia.** Si se es transparente en el proceso de desarrollo, entonces el cliente o el usuario final saben lo que se está haciendo, tienen más confianza en quien desarrolla porque se les dice que se está haciendo y se hace, ya que se les proporciona software cada cierta cantidad de *sprints*, sabrán qué tipo de software están obteniendo y, en consecuencia, proporcionarán una retroalimentación y se mejorara en la siguiente iteración al usar esa retroalimentación en la etapa de planificación.
- **Costos y plazos predecibles.** Como Ágil promueve ciclos de vida de desarrollo a muy corto plazo, la cantidad de variables que están allí cuando se calcula el costo y el cronograma serán menor y, por lo tanto, se podrá predecir el costo y programar todas las tareas de manera mucho más eficiente.

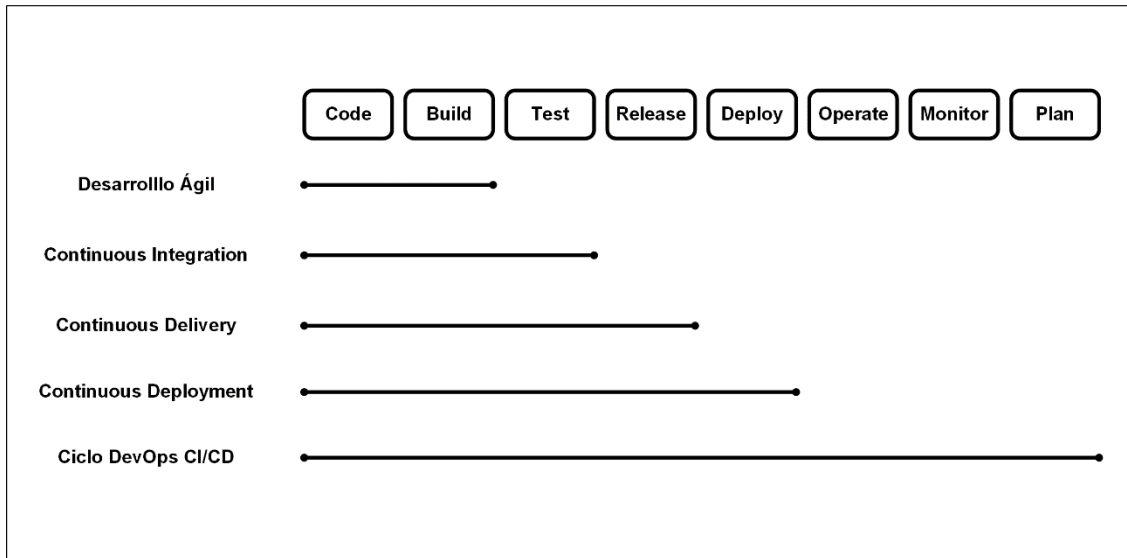
- Permite el cambio. El beneficio principal de Ágil es que permite el cambio. Los valores y principios de los que Ágil habla básicamente promueven el cambio, porque si se es capaz de adaptarse y cambiar de acuerdo con el mercado, entonces se es capaz de aumentar las ganancias del negocio, así como saciar las necesidades de los mercados o satisfacer al cliente o usuario final.
- Mejora la calidad. Ayuda a mejorar la calidad del producto en general, no solo el aspecto de pulido, aumenta la moral de los empleados y del equipo, y de esta manera ellos quieren hacer un mejor producto y, por lo tanto, se hace un mejor producto.

1.5. Niveles de automatización

El objetivo principal del desarrollo de software es construir, desarrollar, probar e implementar una pieza de software, pero en última instancia, tomar cualquier proceso manual y automatizarlo. Por lo que idealmente se debe de automatizar la mayor parte del proceso de construcción e implementación.

Los beneficios de automatizar la mayor parte de estos procesos es que las tasas de despliegue aumentan. Por lo que cuanto más se pueda automatizarlos, significa potencialmente que más y más rápido se podrán desplegar cualquier actualización del software. Por lo tanto, el rendimiento aumenta y se obtiene la idea de facilidad de escalabilidad, flexibilidad y confiabilidad. Estas son todas las características que uno podría conectar a la automatización en el desarrollo de software.

Figura 14. Niveles de automatización



Fuente: elaboración propia, realizado con Microsoft Visio.

1.5.1. Desarrollo Ágil

Desarrollo basado en Ágil, el cual es una mentalidad que permite sortear limitaciones desconocidas.

1.5.2. *Continuous Integration* (Integración continua)

La Integración Continua significa que los equipos utilizan Pruebas Continuas para realizar automáticamente la integración frecuente de pequeños cambios en el repositorio remoto.

1.5.3. *Continuous Testing* (Pruebas continuas)

Lo que significa que los cambios de código se prueban de forma automática y continua en un entorno estandarizado en todas las fases de la canalización de entrega, desde las pruebas locales hasta las pruebas a nivel de sistema en un entorno similar al de producción.

1.5.4. *Continuous Delivery* (Entrega continua)

La Entrega Continua extiende la Integración Continua y las Pruebas Continuas mediante crear automáticamente paquetes de software, entregables que se puedan lanzar bajo demanda y después de que se hayan creado y verificado los cambios en el código a través de la Integración Continua, el objetivo es tener siempre un software verificado y listo para la implementación.

1.5.5. *Continuous Deployment* (Despliegue continuo, Implementación continua)

Proceso en el que el código se despliega automáticamente de forma autónoma en el entorno de producción.

1.5.6. *Continuous Monitoring* (Monitoreo continuo)

Las organizaciones realizan un seguimiento de las métricas y los registros para ver cómo el rendimiento de las aplicaciones y la infraestructura afectan la experiencia del usuario final.

1.5.7. Ciclo DevOps CI/CD

Es un ciclo de vida de desarrollo de software en el que desarrollo y operaciones trabajan juntos para minimizar los problemas, acelerar los procesos y garantizar la calidad al momento de desarrollar, integrar y entregar una pieza de software.

1.6. CI/CD

CI *Continuous Integration* y CD *Continuous Delivery* o *Continuous Deployment*.

Se está hablando de integrar continuamente, entregar continuamente e implementar continuamente una pieza de software, haciendo todo esto de la manera más confiable posible.

Se piensa en CI/CD como una colección de principios y prácticas operativas que, en última instancia, ayuda a desarrollar y entregar cambios de código frecuentes de una manera realmente confiable.

2. DEVOPS

2.1. Definición

La palabra DevOps se acuñó combinando las palabras *development* (desarrollo) y *operations* (operaciones), DevOps es un conjunto de conceptos culturales, prácticas y tecnologías que mejoran la capacidad de una organización para producir aplicaciones y servicios de alta velocidad, lo que permite evolucionar y mejorar los productos más rápido que los métodos tradicionales de desarrollo de software y gestión de infraestructura.

DevOps resuelve el conflicto de comunicación entre los equipos de desarrollo y operaciones. El equipo de desarrollo planifica y escribe código, es quien realmente construye el producto, ese código incluye nuevas características, correcciones de errores, actualizaciones de seguridad y bastantes cosas más. El equipo de operaciones gestiona la infraestructura, se asegura de que los servidores se mantengan activos, sean cómodos y acogedores, lanza el software dado por el equipo de desarrollo, entre otras cosas.

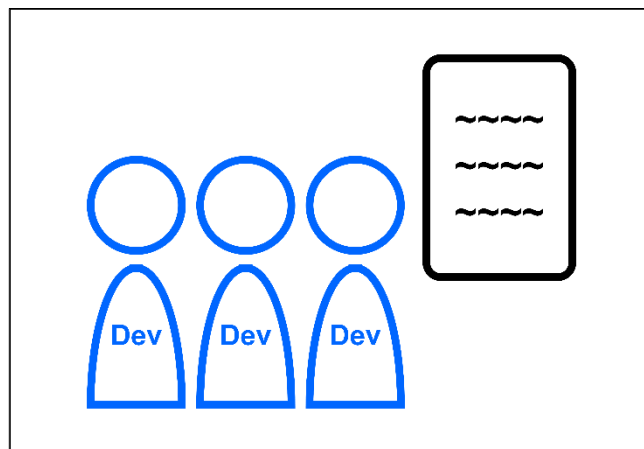
Se piensa en DevOps como una solución, el problema es cómo desarrollar de forma colaborativa, implementar pruebas y escalar de forma segura y fácil, al mismo tiempo que se garantiza una integración y entrega de software sin problemas, por lo que con eso en mente se puede decir que DevOps es la práctica de ingenieros de desarrollo e ingenieros de operaciones que participan juntos en todo el ciclo de desarrollo del software, desde el diseño hasta el proceso de desarrollo y apoyo en producción.

2.2. Desarrollo de software con métodos tradicionales

El proceso de desarrollo de software utilizando métodos tradicionales suele seguir los siguientes pasos:

- Un equipo de desarrollo diseña la aplicación y escribe el código para especificar las acciones de un producto.

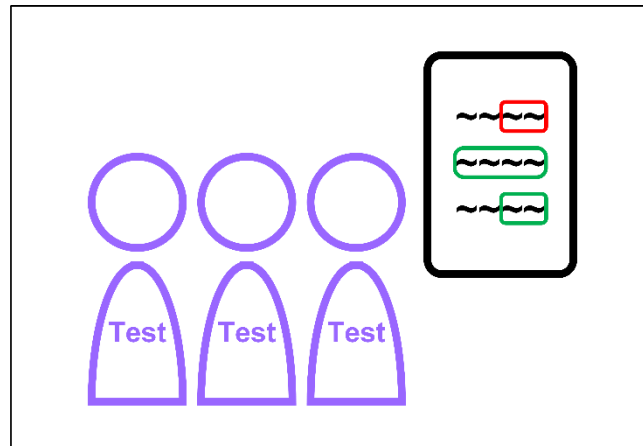
Figura 15. **Desarrollo con métodos tradicionales**



Fuente: elaboración propia, realizado con Microsoft Visio.

- Los ingenieros de control de calidad revisan el código para asegurarse de que la aplicación funcione según lo previsto.

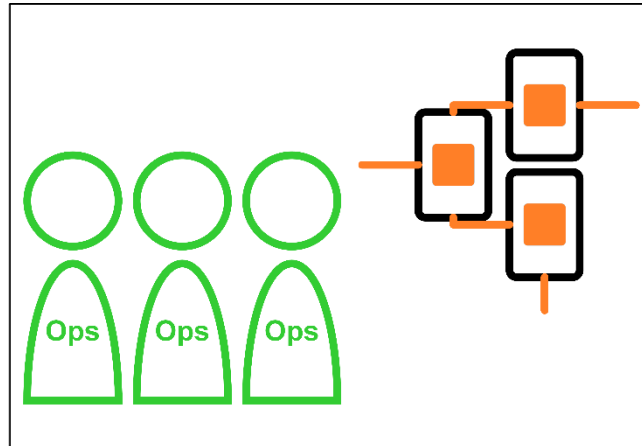
Figura 16. **Control de calidad con métodos tradicionales**



Fuente: elaboración propia, realizado con Microsoft Visio.

- Un equipo de operaciones formado por ingenieros de lanzamiento, administradores de bases de datos y administradores de red configuran el código en uno o varios servidores de producción, se encargan de la seguridad y monitorean el rendimiento de estos. Este equipo es el encargado de poner el producto a disposición de los usuarios finales.

Figura 17. **Operaciones con métodos tradicionales**



Fuente: elaboración propia, realizado con Microsoft Visio.

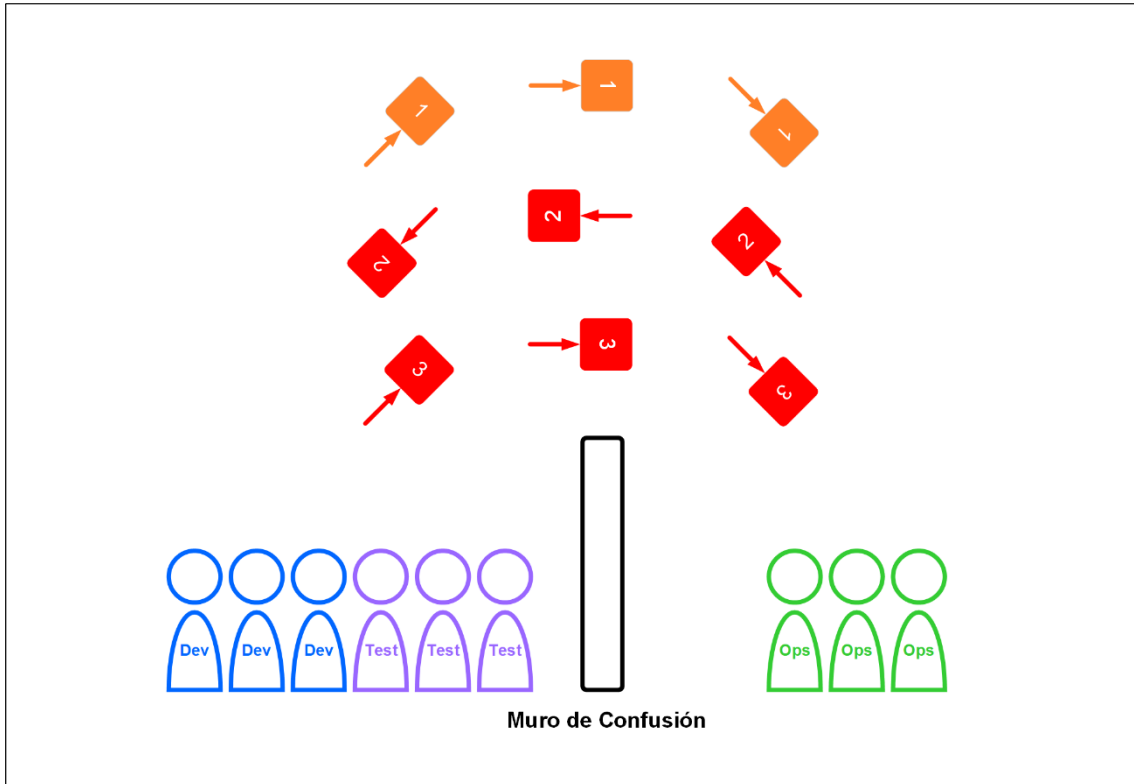
2.2.1. **Conflicto**

No siempre el software que el equipo de desarrollo entrega al equipo de operaciones funciona correctamente en el ambiente de producción, este conflicto entre desarrolladores y operaciones se conoce como el muro de confusión y es común en las organizaciones que siguen las formas tradicionales de desarrollo.

2.3. **Muro de confusión**

Los desarrolladores modifican un producto para aportar valor a los usuarios mientras las operaciones aseguran que el producto sea confiable y estable. Los dos equipos trabajan por separado con diferentes objetivos y entendimientos de lo que es bueno y lo que es malo. Los desarrolladores arrojan el lanzamiento de software por el muro a las operaciones para su despliegue y una vez que ocurre un problema, ambos equipos luchan por definir por qué sucedió.

Figura 18. **Muro de confusión**



Fuente: elaboración propia, realizado con Microsoft Visio.

Para prevenir situaciones tan frustrantes, la comunidad de desarrollo de software ha dado forma a la estrategia llamada DevOps.

2.4. **Ciclo DevOps CI/CD**

Un flujo de trabajo típico de un equipo de DevOps es CI/CD.

DevOps CI/CD reúne a todos los involucrados en el desarrollo y la implementación de software en un único flujo de trabajo altamente automatizado,

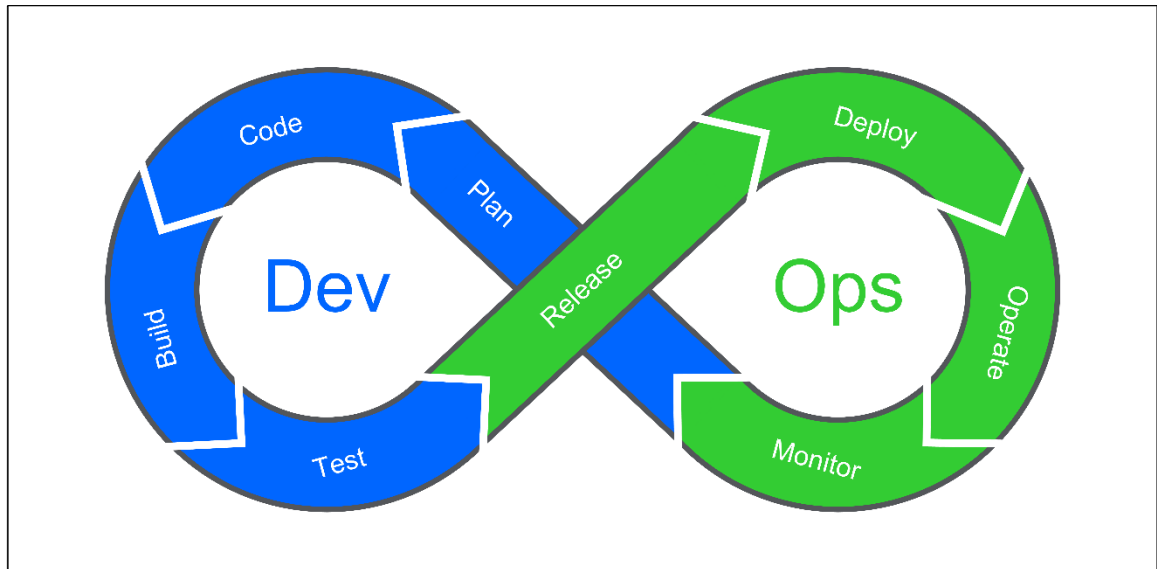
con el único objetivo de implementar rápidamente software de alta calidad, al mismo tiempo que mantiene la integridad y la seguridad del sistema.

DevOps CI/CD hace que con frecuencia los desarrolladores confirmen su código en un repositorio compartido, cada confirmación desencadena un flujo de trabajo automatizado en un servidor de CI/CD que puede notificar a los desarrolladores de cualquier problema al integrar sus cambios.

Un ciclo de vida realmente básico de una aplicación sería en el que se diseña, se construye, se desarrolla, se prueba, se hacen algunas pruebas de aceptación del usuario, de garantía, de calidad y luego se implementa.

Entonces el Ciclo DevOps CI/CD se refiere al ciclo de vida del desarrollo de software. Los pasos generalmente se escriben dentro de un bucle infinito, ya que es un ciclo que se repite para siempre.

Figura 19. **Ciclo DevOps CI/CD**



Fuente: elaboración propia, realizado con Microsoft Visio.

2.4.1. Beneficios

- Automatización. DevOps automatiza cosas que de otro modo los desarrolladores tendrían que hacer manualmente, lo que aumentará la velocidad.
- Velocidad. DevOps permite moverse a una velocidad alta, lo que permite innovar para los consumidores con mayor rapidez, adaptarse mejor a los mercados cambiantes y ser más eficiente en la generación de resultados comerciales.
- Entrega rápida. Con DevOps se puede aumentar el número de lanzamientos.

- Detección de Errores. DevOps detecta pequeños problemas temprano antes de que puedan convertirse en desastres mayores y eso da como resultado una mayor calidad de código.
- Seguridad. Sin sacrificar la seguridad se puede adoptar un paradigma DevOps utilizando estándares de cumplimiento automatizados, controles detallados y enfoques de gestión de configuración.

2.4.2. Pasos

Un sin fin de desarrollo de código, pruebas automatizadas, integración e implementación es lo que es el Ciclo DevOps CI/CD.

Los miembros del equipo completan tareas priorizadas de una iteración corta actual, un *sprint*.

2.4.2.1. Plan (planificación)

Los *stakeholders* y el equipo de desarrollo analizan los objetivos del proyecto y se crea un plan.

Se planean mejoras teniendo en cuenta los comentarios anteriores.

2.4.2.2. Code (codificación)

Los ingenieros de software escriben el código.

Usan herramientas de administración de código fuente como Git.

2.4.2.3. Build (construcción)

Los ingenieros de software construyen o compilan el código en pequeños trozos varias veces al día.

Usan herramientas como Maven y Gradle como una forma de construir consistentemente su entorno.

2.4.2.4. Test (pruebas)

Los ingenieros de control de calidad prueban el código recién construido con herramientas de automatización como Watir, *Selenium*, *Micro Focus UFT One*, entre otros. Si la herramienta encuentra algún error, el código con errores se envía de vuelta a los desarrolladores para correcciones.

Los miembros del equipo monitorean los cambios realizados en el código con los sistemas de control de versiones para detectar problemas de integración de antemano.

2.4.2.5. Release (lanzamiento o liberación)

El código que pasa las pruebas automatizadas se integra en un repositorio compartido en un servidor, el código se envía a aseguramiento de la calidad (QA, *quality assurance*) donde un pequeño porcentaje de usuarios lo prueba.

2.4.2.6. Deploy (despliegue)

Si el equipo no encuentra ningún problema, despliega el código en un servidor público es decir en un servidor de producción.

2.4.2.7. Operate (operación)

Desplegado en el servidor de producción es usado por los usuarios finales o clientes.

Se espera que los cambios no afecten la estabilidad del sistema.

Se utilizan herramientas como Ansible y Chef, que operan y administran entornos de producción. De modo que cuando el código se implemente en los diferentes servidores de producción, sea compatible con el entorno y se tengan los resultados esperados.

2.4.2.8. Monitor (monitoreo)

El equipo supervisa cómo está funcionando, monitorea y recopila comentarios de usuarios y colegas.





2.5. Efectividad

DevOps no es una herramienta que se pueda comprar y comenzar a usar de inmediato, no hay un antes y un después distintos, además de los cambios en el proceso y las herramientas, DevOps significa cambiar la mentalidad del equipo.

Accelerate State of DevOps 2021 es una investigación anual de las empresas que siguen este camino, se miden con respecto a cuatro métricas de rendimiento clave: tiempo de espera para los cambios, frecuencia de despliegue, tiempo medio para restaurar el servicio y tasa de cambio fallido; el porcentaje de cambio lleva a un servicio degradado y que más tarde debe ser corregido. Las organizaciones de mayor rendimiento despliegan código 973 veces más

frecuentemente que las de bajo rendimiento y pasan de código confirmado a desplegado con éxito 6570 veces más rápido, estas empresas también logran 6570 veces más rápida recuperación de incidentes y tienen una tasa de fallas de cambio 3 veces menor que las empresas de bajo rendimiento.

Tabla I. Métricas de rendimiento clave

Software delivery performance metric	Elite	High	Medium	Low
 Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months	Fewer than once per six months
 Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one hour	Between one day and one week	Between one month and six months	More than six months
 Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day	Between one day and one week	More than six months
 Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0%-15%	16%-30%	16%-30%	16%-30%

Fuente: Google Cloud (2021). *Accelerate State of DevOps 2021*.

Figura 20. Organizaciones de mayor rendimiento



Fuente: Google Cloud (2021). *Accelerate State of DevOps 2021*.

3. USER EXPERIENCE

3.1. Definición

User Experience (Experiencia de Usuario) es la experiencia que alguien puede tener al interactuar con un producto.

Hoy en día, este término se usa con mayor frecuencia en un contexto de productos digitales: aplicaciones móviles, sitios web, juegos, entre otros, pero UX es un término mucho más amplio y puede usarse para describir cualquier interacción entre el usuario y el producto que pueda ir mucho más allá del espacio digital.

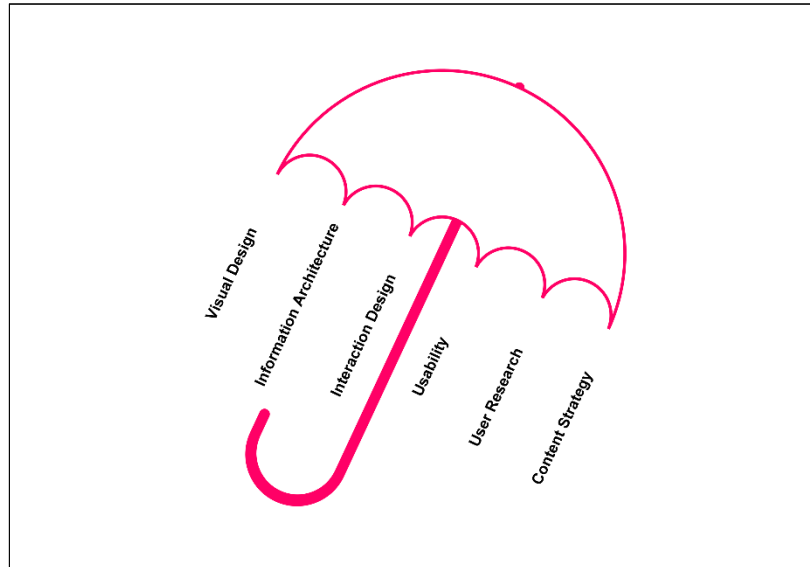
3.2. Diseño de UX

Se centra en todos los aspectos relacionados con hacer la interacción entre el usuario y el producto. El diseño de UX encapsula una amplia gama de otros términos relacionados con la creación, el consumo o el uso del producto, como interfaz de usuario, marca, usabilidad, funciones, contenido, entre otros.

3.2.1. UX Umbrella (Paraguas UX)

Existen variaciones de las piezas que se utilizan para construir un producto digital típico, pero en general estas se encuentran contenidas en lo que se conoce como *UX Umbrella* (Paraguas UX).

Figura 21. **UX Umbrella**



Fuente: elaboración propia, realizado con Microsoft Visio.

3.2.1.1. *Visual design* (diseño visual)

El diseño visual en este contexto se refiere a la *user interface* (UI, interfaz de usuario), que, en términos simples, define cómo se ven los productos, es una capa visual de la UX. Consta de todos los componentes con los que el usuario puede interactuar. Por ejemplo, lo que se ve en pantalla al usar una aplicación.

3.2.1.2. *Information architecture* (arquitectura de la información)

La arquitectura de la información es una estructura de cómo los usuarios pueden acceder a las diferentes funciones o contenidos de los productos. Es una base de dónde deben ir en la pantalla las diferentes piezas de la interfaz.

Un ejemplo típico sería la navegación de cualquier aplicación, donde se presenta una lista de diferentes opciones, cuántos elementos hay en la lista y a dónde llevan dentro de la aplicación.

Esencialmente es el arte y la ciencia de organizar y etiquetar las cosas dentro del producto.

3.2.1.3. *Interaction design* (diseño de interacción)

El diseño de interacción, como su nombre indica, se refiere al diseño de interacciones con productos y usuarios. Se centra específicamente en un momento de esas interacciones e involucra 5 dimensiones diferentes: palabras, representaciones visuales, objetos o espacios físicos, tiempo y comportamiento.

3.2.1.3.1. Palabras

Las palabras son textos que proporcionan la cantidad correcta de información a los usuarios, deben ser fáciles de entender y deben comunicar la información de manera sucinta. Generalmente se utilizan en etiquetas de botones.

3.2.1.3.2. Representaciones visuales

Las representaciones visuales son elementos gráficos como imágenes, tipografía o iconos con los que interactúa el usuario. Suelen complementar a las palabras para comunicar información a los usuarios.

3.2.1.3.3. Objetos o espacios físicos

Objetos o espacios físicos se refiere al medio a través del cual los usuarios pueden interactuar con el producto. Por ejemplo, una computadora portátil a través del teclado y el panel o la pantalla táctiles a través de los dedos.

3.2.1.3.4. Tiempo

El tiempo se refiere a los cambios de estado que se dan con el paso del tiempo, como las animaciones o las notificaciones.

3.2.1.3.5. Comportamiento

El comportamiento se refiere a cómo el producto reacciona a las acciones de los usuarios y les proporciona retroalimentación.

3.2.1.4. Usability (usabilidad)

La usabilidad evalúa la facilidad de uso de las interfaces de usuario, pero también se refiere a diferentes métodos para mejorar la facilidad de uso de las interfaces durante el proceso de diseño.

Según el Nielsen (s.f.) un gurú de la usabilidad expresa que la usabilidad se puede definir por 5 componentes de calidad: *learnability* (capacidad de aprendizaje), *efficiency* (eficiencia), *memorability* (hecho memorable), *errors* (errores) y *satisfaction* (satisfacción).

3.2.1.4.1. *Learnability* (capacidad de aprendizaje)

La capacidad de aprendizaje determina qué tan fácil es para los usuarios realizar tareas básicas cuando interactúan con los productos por primera vez.

3.2.1.4.2. *Efficiency* (eficiencia)

La eficiencia se refiere a la rapidez con la que después del período de aprendizaje inicial los usuarios pueden realizar tareas básicas.

3.2.1.4.3. *Memorability* (hecho memorable)

El hecho memorable define qué tan fácil es usar el producto después de un período de no usarlo.

3.2.1.4.4. *Errors* (errores)

Los errores se refieren a ocurrencias de errores, con qué frecuencia los usuarios los encuentran o causan y qué tan fácil (o no) pueden recuperarse de ellos mientras interactúan con el producto.

3.2.1.4.5. *Satisfaction* (satisfacción)

La satisfacción define qué tan agradable es la interacción con el producto.

3.2.1.5. **User research (investigación de usuarios)**

La investigación de usuarios se refiere al estudio de los grupos de usuarios objetivo de los productos, sus necesidades y puntos débiles.

A través de varios métodos, *frameworks* y prácticas, los investigadores de UX tienen como objetivo descubrir los problemas con los que están lidiando y las oportunidades para utilizar en el proceso de diseño.

Hay dos aspectos de la investigación de usuarios: cualitativo y cuantitativo.

- El cualitativo tiene como objetivo ayudar a crear una comprensión profunda de por qué los usuarios se comportan de la manera en que lo hacen.
- El cuantitativo es lo que los usuarios realmente hacen, se prueban las suposiciones hechas después de la investigación cualitativa.

También se puede considerar la investigación de usuarios desde los enfoques actitudinal y conductual.

- En el actitudinal se escucha lo que dicen los usuarios a través de entrevistas.
- En el conductual se estudian las acciones de los usuarios a través de la observación.

La mejor comprensión de las necesidades y los puntos débiles de los usuarios se puede lograr cuando se utiliza la investigación cualitativa y cuantitativa con la combinación de ambos enfoques, el actitudinal y el conductual.

3.2.1.6. *Content strategy* (estrategia de contenido)

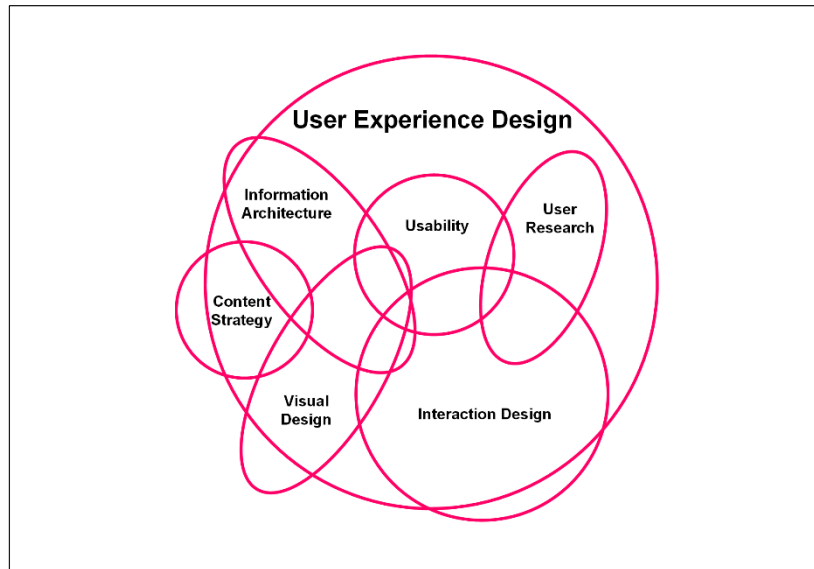
Halvorson (2008) lo describió como “Planes de estrategia de contenido para la creación, publicación y gobernanza de contenido útil y usable” (párr. 2).

Los videos, los textos, las imágenes, los sonidos y otros elementos tienen el propósito de lograr los objetivos comerciales y conectarlos con las necesidades de los usuarios.

Todos los productos digitales dependen del contenido, y para muchos de ellos, es su principal valor para los usuarios. Una estrategia de contenido bien definida y ejecutada garantiza que el producto ofrezca una gran Experiencia de Usuario.

Muchas de las piezas de diseño de UX mencionadas se superponen entre sí por márgenes más pequeños o grandes.

Figura 22. **Piezas de diseño de Experiencia de Usuario**



Fuente: elaboración propia, realizado con Microsoft Visio.

En organizaciones y productos más grandes, esas piezas individuales podrían conducir a la especialización de diseñadores de UX individuales o incluso de departamentos de diseño completos, por lo que podría haber investigadores de UX dedicados, especialistas en usabilidad, diseñadores de interacción, diseñadores visuales, entre otros. En organizaciones más pequeñas, todas esas piezas suelen ser manejadas de manera holística por diseñadores de UX.

3.3. Proceso de diseño de UX

Un proceso de diseño da estructura y orientación sobre cómo impulsar el proyecto, especialmente cuando se queda atascado y no se sabe qué hacer a continuación, apoyarse en un proceso será muy crítico.

No existe un proceso de diseño de UX correcto o perfecto porque cada proyecto, equipo y empresa son diferentes. Por lo tanto, se tiene que ajustar el proceso para hacer lo que tenga más sentido para el integrante del equipo y el equipo.

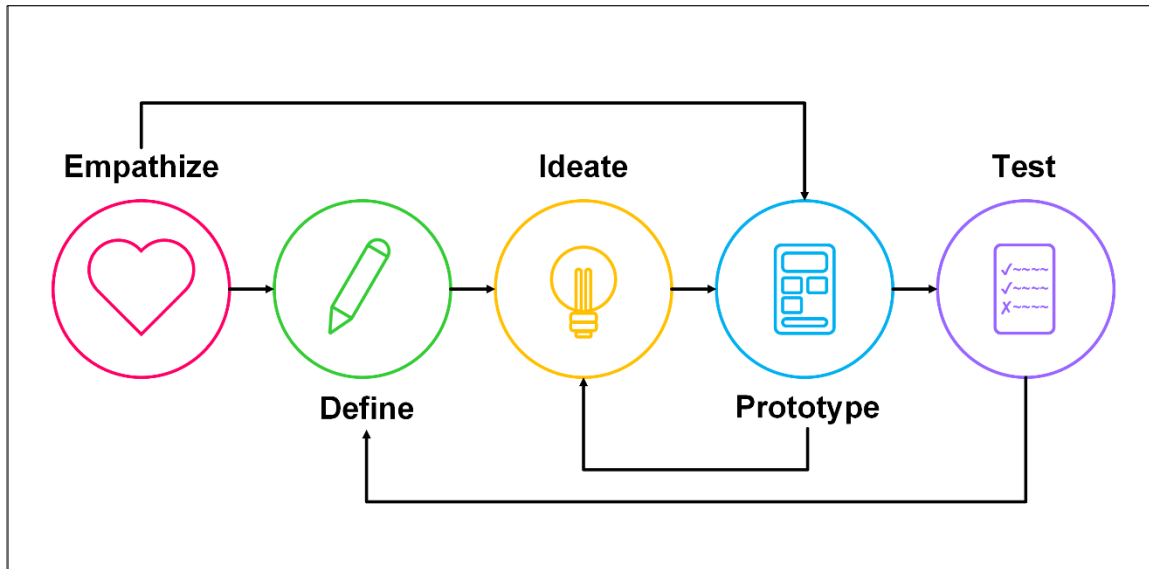
3.3.1. *Design Thinking* (Pensamiento de Diseño)

Es una filosofía y un conjunto de herramientas para ayudar a resolver problemas creativamente.

Mira toda la resolución creativa de problemas a través de la lente del diseño centrado en el ser humano. Se trata de empatizar y descubrir para quien se está diseñando y cuáles son sus necesidades, para luego resolver el problema innovando en función de esas necesidades.

Existen cinco pasos principales para cualquier proceso de *Design Thinking*. Los ejercicios exactos dentro de cada paso no están definidos y el proceso específico es muy flexible. Cada empresa, diseñador y *design thinker* (pensador de diseño) lo hace de manera diferente, pero los cinco pasos siempre siguen siendo los mismos.

Figura 23. **Design Thinking**



Fuente: elaboración propia, realizado con Microsoft Visio.

3.3.1.1. **Empathize (empatizar)**

Se trata básicamente de comprender a las personas para las que se diseña el producto o el servicio.

La mayor parte del tiempo se pasa recopilando información. Se hacen entrevistas a usuarios tratando de averiguar ¿para quién se está haciendo esto?, ¿cuál es su problema? y ¿qué hacen estas personas?

3.3.1.2. **Define (definir)**

Se toma todo lo aprendido en el paso anterior empatizar y se convierte en información que realmente se pueda usar.

Se llegan a conclusiones sobre necesidades, problemas, desafíos y observaciones.

3.3.1.3. *Ideate (idear)*

Se toma todo lo aprendido en los dos pasos anteriores y se ocurren ideas que son soluciones.

3.3.1.4. *Prototype (prototipo)*

Se seleccionan las ideas que se cree podrían valer la pena convertirlas en algo más realista y se convierten en prototipos simples y comprobables.

Estos prototipos no están completamente diseñados, no están completamente codificados, no están completamente convertidos en algo real, pero son en esencia una fachada, esencialmente algo falso, ya sea un producto digital o un producto físico, solo se está tratando de hacer rápidamente algo que se pueda probar con usuarios reales.

3.3.1.5. *Test (prueba)*

Se toman los prototipos y se prueban con personas reales y estas personas reales son seleccionadas en función de lo aprendido en el paso uno empatizar y esencialmente estas personas reales usaran el producto y se obtendrá retroalimentación de ellos en tiempo real.

Una vez que se tienen los resultados de la prueba habrá muchos nuevos conocimientos, por lo que estos conocimientos se reúnen y se vuelve a la fase de definición, luego a la fase de idear, que en función de los nuevos aprendizajes

ocurrirán ideas nuevas, ideas actualizadas, por lo que se va a la fase de prototipar y luego a la fase de probar y ese es el ciclo central de *Design Thinking*.

3.3.2. Lean UX

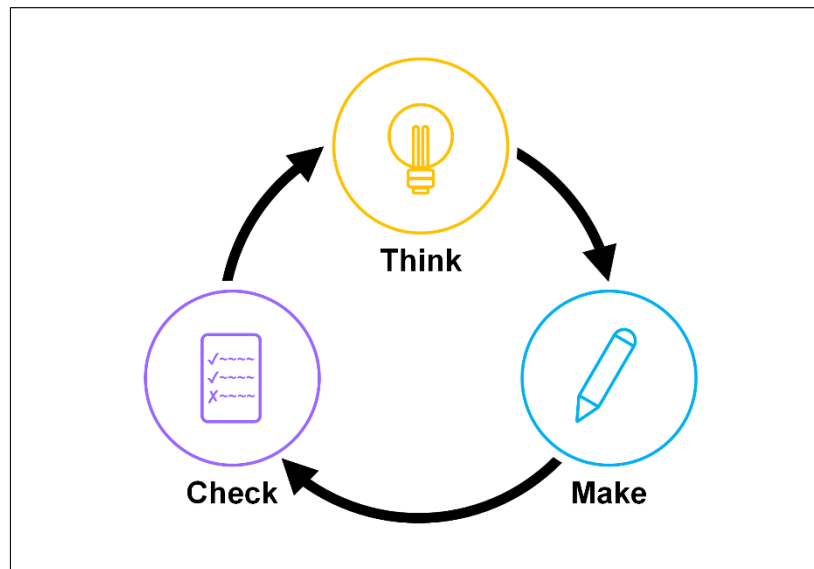
Lean UX es un *framework* de diseño que se enfoca en aprender lo más rápido posible y reducir el tiempo del ciclo.

La idea es que en lugar de solo asumir que se sabe exactamente lo que se debe construir, se forme una hipótesis de algo que podría funcionar y luego se encuentre la manera más rápida de validar o invalidar esa hipótesis.

Lean UX es una forma de aplicar los principios de *Lean Startup* a la práctica de diseño de Experiencia de Usuario. *Lean UX* se apoya en tres pilares clave: *Lean Startup*, *Design Thinking* y *Agile Development*.

Lean UX se da en un ciclo de tres pasos *think* (pensar), *make* (hacer) y *check* (comprobar).

Figura 24. *Lean UX*



Fuente: elaboración propia, realizado con Microsoft Visio.

3.3.2.1. *Think* (pensar)

Se investiga, se recopilan datos, se analizan datos, se realiza una lluvia de ideas y se llega a una hipótesis.

La hipótesis debe contener cuatro elementos clave:

- Función que se va a construir
- Usuarios a los que se dirige
- Resultado que se está buscando
- ¿Cómo se va a medir el éxito?, ¿Cómo se va a reconocer el éxito?

3.3.2.2. Make (hacer)

Una vez que se tiene la hipótesis, se construye un MVP (*Minimum Viable Product*). Un MVP es un Producto Mínimo Viable, es lo más pequeño posible que se puede construir, que ayudará a saber si la hipótesis es cierta o no.

3.3.2.3. Check (comprobar)

Se lanza la nueva función y se empieza a medir usando uno o varios métodos y técnicas como como *A/B testing*, encuestas, *usability testing*, entre otros. Se determina si la función es lo que se esperaba, si necesita mejoras o si debe de abandonarse por completo y comenzar de nuevo.

3.3.3. Design Sprint 2.0 (Sprint de Diseño 2.0)

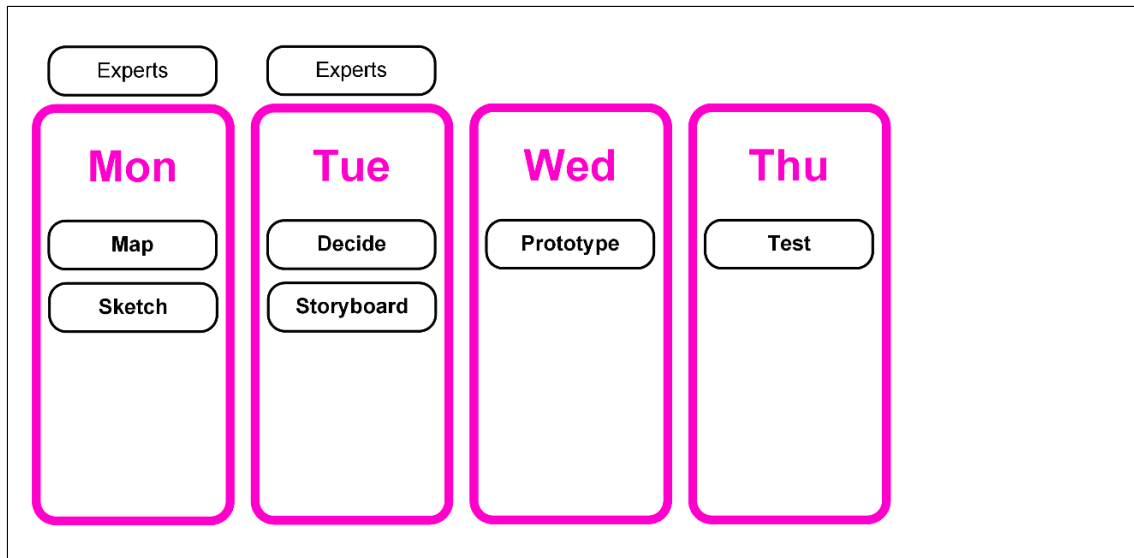
Design Sprint creado en Google Ventures por Jake Knapp en 2010 y tiempo después compartido en su libro *Sprint* en 2016.

Design Sprint es un proceso para resolver problemas muy grandes y probarlos en un corto período de tiempo. Puede durar tres, cuatro o cinco días, dependiendo del formato que se utilice. Sin importar el formato, todos siguen el mismo conjunto de ejercicios para producir resultados consistentes orientados a la acción.

Los equipos de *Design Sprint* tienen siete personas como máximo, más un facilitador. Generalmente incluyen miembros del equipo que serán responsables de la implementación o aquellos que llevan información pertinente o tienen habilidades para la toma de decisiones.

Design Sprint 2.0 es un formato de cuatro días creado por Jake Knapp y AJ&Smart, diferente del formato original de cinco días.

Figura 25. ***Design Sprint 2.0***



Fuente: elaboración propia, realizado con Microsoft Visio.

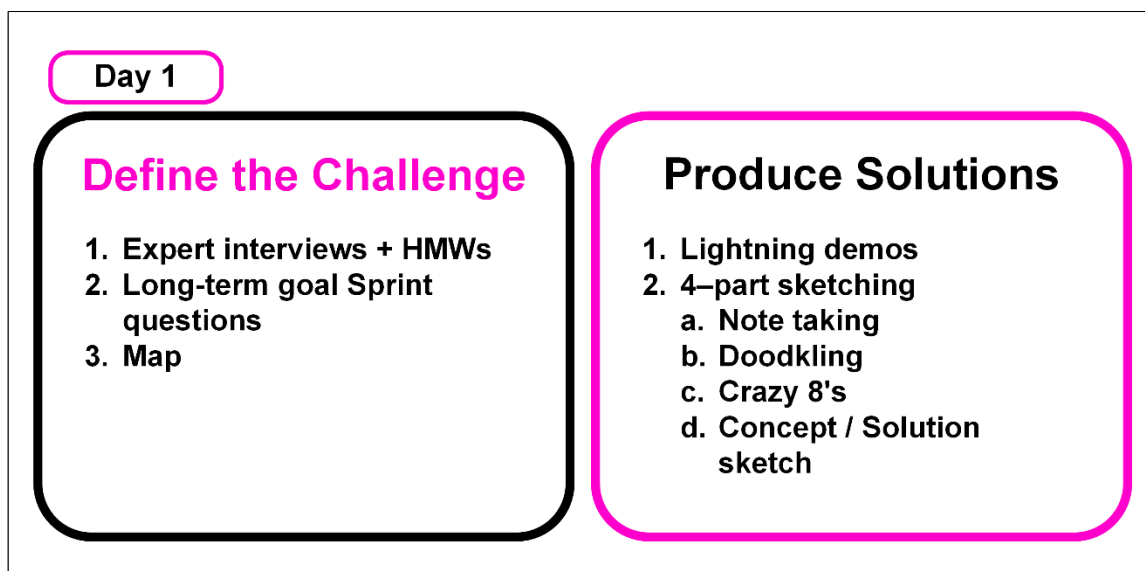
El lunes y el martes el equipo y los expertos están en la sala. Los expertos son todos los que tienen información significativa para contribuir y son responsables de las decisiones que se toman. Al final del martes los expertos abandonan la sala. El miércoles y el jueves solo el equipo está en la sala.

3.3.3.1. Lunes

El equipo invita a expertos del servicio o del producto, generalmente *product owners*, CEOs (*Chief Executive Officers*, directores ejecutivos) y altos ejecutivos.

El lunes los ejercicios incluyen mapear problemas, definir el desafío y esbozar. La primera mitad del día se dedica a la definición del desafío y la segunda mitad del día se dedica a la producción de soluciones.

Figura 26. **Lunes *Design Sprint 2.0***



Fuente: elaboración propia, realizado con Microsoft Visio.

3.3.3.1.1. ***Define the challenge*** (definir el desafío)

Los ejercicios de definir el desafío incluyen:

- *Expert interviews + HMWs (How Might We ...?)* (entrevistas a expertos + ¿cómo podríamos ...?)

Una entrevista a un experto toma aproximadamente 10 minutos. Se tienen a los *stakeholders* clave, todos con información relevante que brindan debates y

hablan sobre los problemas, los desafíos y todo lo que tienen en mente sobre el producto o el servicio específico.

Todos los demás que no están hablando, escriben notas de ¿cómo se podría ... ?, por ejemplo ¿cómo se podría mejorar la experiencia del cliente con ... ?, que representan problemas planteados como oportunidades para enfocarse en el *sprint*. No se está enfocando en la solución, se está enfocando en la oportunidad.

- *Long-term goal + Sprint questions* (objetivo a largo plazo + preguntas del sprint).

El objetivo a largo plazo es optimista, un objetivo en un mundo ideal donde se quiere estar en dos años si todo salió absolutamente perfecto.

Las preguntas de *sprint* son más pesimistas, como, ¿qué se interpondría en nuestro camino?, ¿cuáles son las razones por las que se fallaría?

- *Map* (mapa).

Aquí realmente se está mapeando todo el *workflow* (flujo de trabajo) y el proceso de cómo funciona y opera un negocio, dónde caen los desafíos específicos y quiénes son los actores clave dentro de ese proceso.

3.3.3.1.2. *Produce solutions* (producir soluciones)

La segunda mitad del lunes es producir soluciones.

- *Lightning demos* (demostraciones relámpago).

Las demostraciones relámpago son una forma para que los miembros del equipo presenten inspiraciones, casos de estudio, competidores, cosas que ven en el mundo real que se están haciendo muy bien y se quiere que el equipo se alinee en torno a eso.

- *4-part sketching* (bocetos de 4 partes)

El boceto se hace en 4 pasos:

- *Note taking* (toma de notas)

Comienzan con una parte de toma de notas rápida, tomar notas es solo un ejercicio para que todos comiencen a anotar ideas que quieren esbozar, en las que quieren enfocarse, tal vez vieron mucha inspiración en las demostraciones relámpago que quieren alinearse con eso y prepararse.

- *Doodling* (hacer garabatos)

Es solo un ejercicio de boceto de calentamiento, se podría comenzar a dibujar cajas, componentes, preguntas, cosas como esas.

- *Crazy 8's* (ochos locos).

Durante ocho minutos cada minuto se esbozará algo diferente, ocho cosas diferentes, de modo que cada minuto se trabaja en un *sketch* (boceto) y luego se pasa al siguiente. Se itera rápidamente, enfocándose en diferentes ideas.

- *Concept / Solution sketch* (concepto / boceto de la solución).

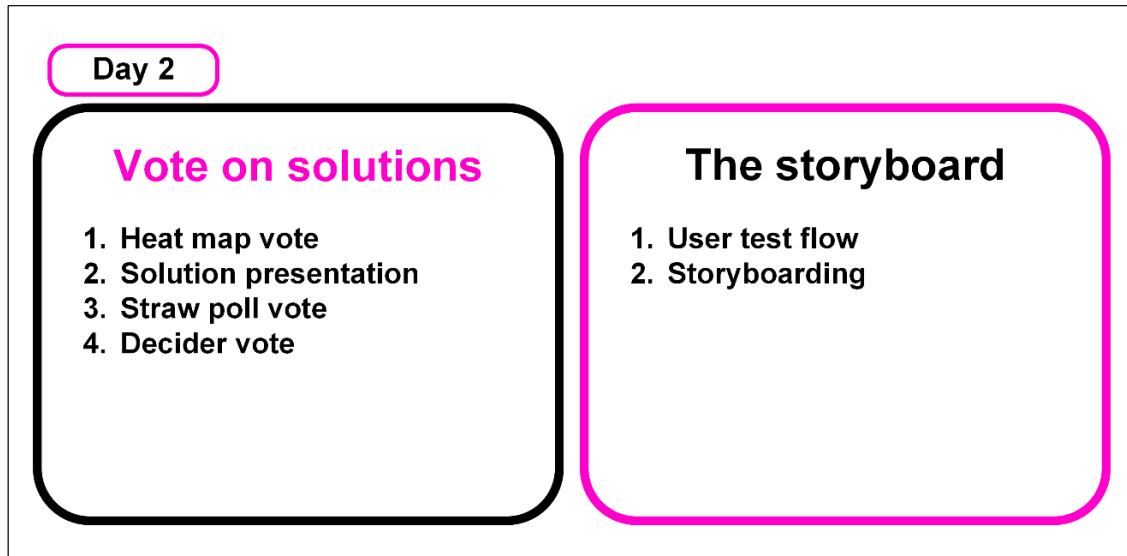
Aquí es donde se toma cualquiera de los *sketches* (bocetos), se vuelve a esbozar en cuatro *frames* (cuadros) aproximadamente que actuarán como un concepto de alma, por lo que todos son un flujo de boceto lineal y luego estos se colgarán y se votarán al día siguiente.

3.3.3.2. Martes

El martes se tiene decisión y *storyboarding* (guion gráfico), para votar sobre las soluciones por seguir y, de hecho, se asigna a lo que actuará como la guía para el prototipo. La primera mitad del día se dedica a votar por las soluciones y la segunda mitad del día se dedica a trabajar en el *storyboarding*.

Al final del día los expertos dejan la habitación y el resto de los días permanecerá solo el equipo.

Figura 27. **Martes Design Sprint 2.0**



Fuente: elaboración propia, realizado con Microsoft Visio.

3.3.3.2.1. ***Vote on solutions*** (votar soluciones)

La primera mitad del martes es votar soluciones.

- *Heat map vote* (votación del mapa de calor)

Es el proceso en el que cada compañero del equipo camina alrededor de los bocetos y coloca calcomanías alrededor de las cosas que le gustan, las cosas que quiere avanzar con características específicas, tal vez sea una redacción específica, tal vez sea todo el concepto en sí mismo y así se obtiene un montón de pequeños votos esparcidos alrededor del *heat mapping* (mapeo de calor) en los bocetos.

- *Solution presentation* (presentación de solución)

Ahora no es el creador el que presenta su boceto, es el moderador o facilitador que presenta el trabajo, porque los bocetos están destinados a ser más anónimos, lo que ayuda a reducir los sesgos y cosas por el estilo. Para que el facilitador pueda leer estos bocetos porque están hechos de una manera muy descriptiva, tienen un título claro, tienen anotaciones y así sucesivamente.

Una vez que se completan las presentaciones, los creadores pueden hablar y completar los espacios en blanco que se tengan sobre el boceto.

- *Straw poll vote* (votación de encuesta de paja)

Cada miembro del equipo pone un único voto en el concepto de su elección.

- *Decider vote* (voto decisivo)

El decisor vota, tomando así la decisión final. Este tiene dos votos y estos dos votos son los únicos que se toman en cuenta y que deciden el concepto para seguir adelante.

Se ha decidido un boceto que puede ir de varias maneras diferentes, puede haber un claro ganador, puede haber dos ganadores diferentes que se combinan o dos ganadores diferentes que se prueban contra sí mismos en los siguientes días.

3.3.3.2.2. *The storyboard* (guion gráfico)

La segunda mitad del martes es el guion gráfico.

- *User test flow* (flujo de pruebas de usuario)

Comienza con ¿cuál es el flujo de los usuarios?, el flujo que se desea probar generalmente comienza con el primer punto de entrada a una experiencia, este podría ser ver un anuncio o leer un artículo que se vincula a una página de *marketing*, por ejemplo.

El flujo final sería el objetivo, el resultado final que se desea que el usuario alcance, lo que luego se vincularía al objetivo de dos años.

- *Storyboarding* (guion gráfico)

Después de que se haya definido el flujo, simplemente se toma cada paso específico del flujo, se mapea y se esboza en *frames* (cuadros) que serán los *lo-fi sketches* (bocetos de baja fidelidad) para dirigirse al *high fidelity prototype* (prototipo de alta fidelidad).

3.3.3.3. Miércoles

Se pasa el día completo creando prototipos, el *test script* (guion de prueba) y el *test outline* (esquema de prueba), para que el prototipo coincida directamente con el *storyboard* (guion gráfico) y la *test flow* (prueba de flujo) que se hizo en la segunda mitad del día anterior y hacer que funcione y tenga todas las

interacciones necesarias para cualquier información que se esté tratando de obtener de la *test* (prueba) en el día siguiente.

3.3.3.4. Jueves

El jueves es el *test* (prueba). Se prueba el prototipo en cinco entrevistas individuales (se tienen 5 + 1 usuarios para las entrevistas, el + 1 es por si algún usuario queda mal y no asiste a su entrevista). En el *test* están todos en la sala observando en vivo y anotando las ideas clave positivas y negativas que surgen de interacciones específicas.

Específicamente para observar y tomar notas en el *test*, hay un pequeño formato que implica colocar tareas clave específicas en el flujo que se desea validar, luego los nombres de los participantes en la parte superior y tener una cuadrícula dentro donde se colocan ideas positivas o negativas para cada paso específico.

Tabla II. **Jueves formato observar y tomar notas *Design Sprint 2.0***

	Participant 1	Participant 2	Participant 3	Participant 4	Participant 5	Participant 6
Overall Impression						
Ease of use						
Assign to driver						
Comprehension of job status						
Alerts & messaging						

Fuente: elaboración propia, realizado con Microsoft Visio.

3.4. Perfiles de proceso de diseño de UX

El proceso de desarrollo de software difiere de una empresa a otra, por lo que la cantidad de habilidades, personas y pasos requeridos siempre cambiará.

Sin embargo, se pueden identificar dos patrones generales para los procesos de diseño de UX que dependen del perfil de la empresa.

El primer perfil es el de una empresa que crea software para otras empresas y el segundo perfil es el de una empresa con su propio producto o productos.

3.4.1. Perfil *software houses* (casas de software)

Empresa que crea software para otras empresas, usualmente llamadas *software houses*.

Por lo general, el proceso de desarrollo de software comienza con un cliente que se acerca a la empresa para hablar sobre el producto que desea desarrollar y termina en el momento en que el cliente está satisfecho con el resultado y se hace cargo del producto después de una entrega exitosa. En este caso, el proceso de diseño de UX tiende a ser lineal con un principio y un final definidos.

Las etapas seguidas en el proceso de diseño de UX son las siguientes:

- Se aprende sobre los usuarios, sus puntos débiles y sus necesidades
- A partir del conocimiento reunido, se definen problemas y objetivos
- Se hace una lluvia de ideas sobre diferentes soluciones
- Se hacen prototipos de las distintas soluciones
- El prototipo se pone a prueba con el grupo de usuarios al que va dirigido.

Durante todas las etapas, hay muchas actividades diferentes que los diseñadores de UX realizan para lograr los objetivos finales. Puede haber algunos movimientos de ida y vuelta entre las etapas, pero el factor limitante en forma de presupuesto y tiempo generalmente los restringe.

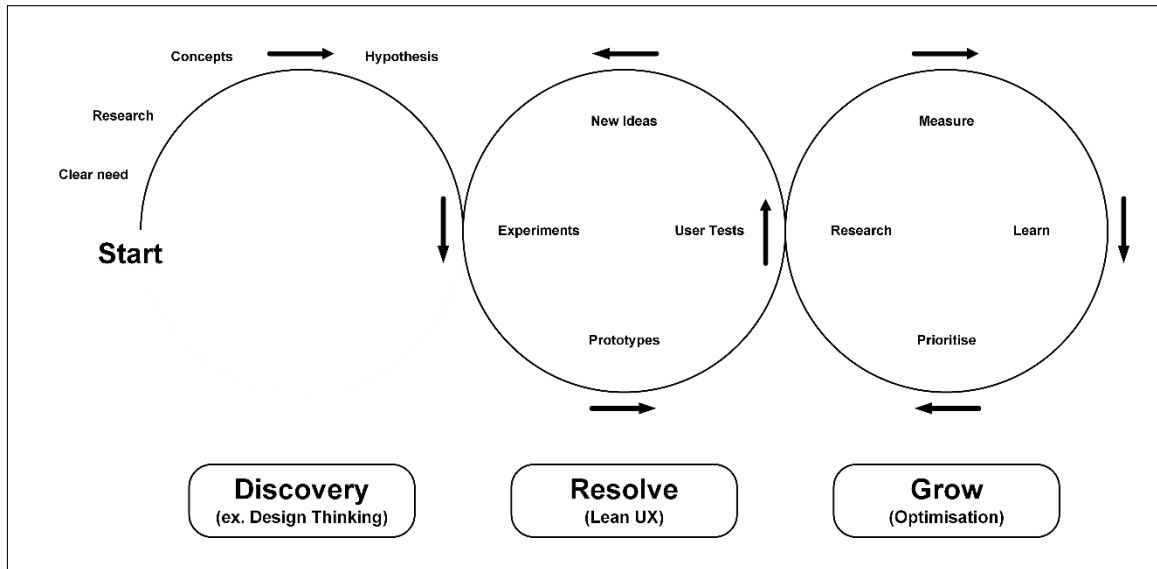
Las etapas y procesos mencionados se describen como *Design Thinking* (Pensamiento de Diseño).

3.4.2. Perfil software propio

Empresa con su propio producto o productos, la mayoría de las veces es una empresa de *Software as a Service* (SaaS, software como servicio).

En este caso, el proceso de diseño UX es continuo sin un principio y un final definidos, aparte de la situación en la que la empresa quiere desarrollar algo nuevo o deja de trabajar en un producto existente.

Figura 28. Perfil software propio



Fuente: elaboración propia, realizado con Microsoft Visio.

Generalmente se centra en el bucle *build-measure-learn* (construir-medir-aprender) precedido por la fase de *discovery* (descubrimiento) o por una necesidad clara. El enfoque de *Design Thinking* es totalmente válido en este caso y se puede utilizar con éxito, aunque la mayoría de las veces los objetivos del proyecto son diferentes y se centran en el refinamiento y la mejora continua del producto.

El proceso comienza de manera similar al *Design Thinking* con una etapa dedicada a comprender el problema, el estado actual del producto y dónde quiere estar la empresa o qué quiere lograr.

Luego, los esfuerzos del equipo se centran en recopilar requisitos (tecnología, auditorías, *stakeholders* (partes interesadas) y usuarios), después de lo cual se definen las soluciones.

La siguiente etapa está dedicada al diseño de prototipos, pruebas y validación en bucle hasta que se logra el resultado deseado y el equipo procede a la implementación.

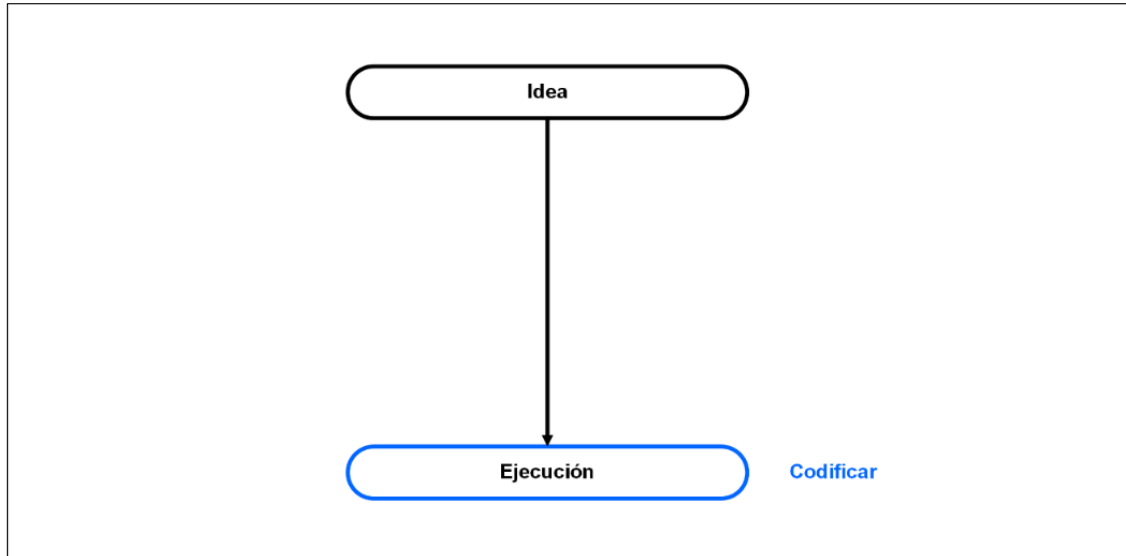
4. DISEÑO DE UX Y DESARROLLO DE SOFTWARE

4.1. Necesidad del diseño de UX en el desarrollo de software

Para desarrollar un producto como lo puede ser un software se necesita comprender las necesidades del usuario y su uso.

Los desarrolladores o los *stakeholders* (partes interesadas) podrían simplemente establecer una solución. Porque tienen alguna corazonada, un presentimiento, tal vez solo vieron a un competidor que ya hace eso y pensaron que también funcionaría para ellos. Construir la solución y entregar como resultado un producto que no se esperaba que se construyera de cierta manera, así que se envía de vuelta y los desarrolladores de software tienen que rehacer todo. Y eso solo crea una gran pérdida de tiempo, esfuerzo y trabajo.

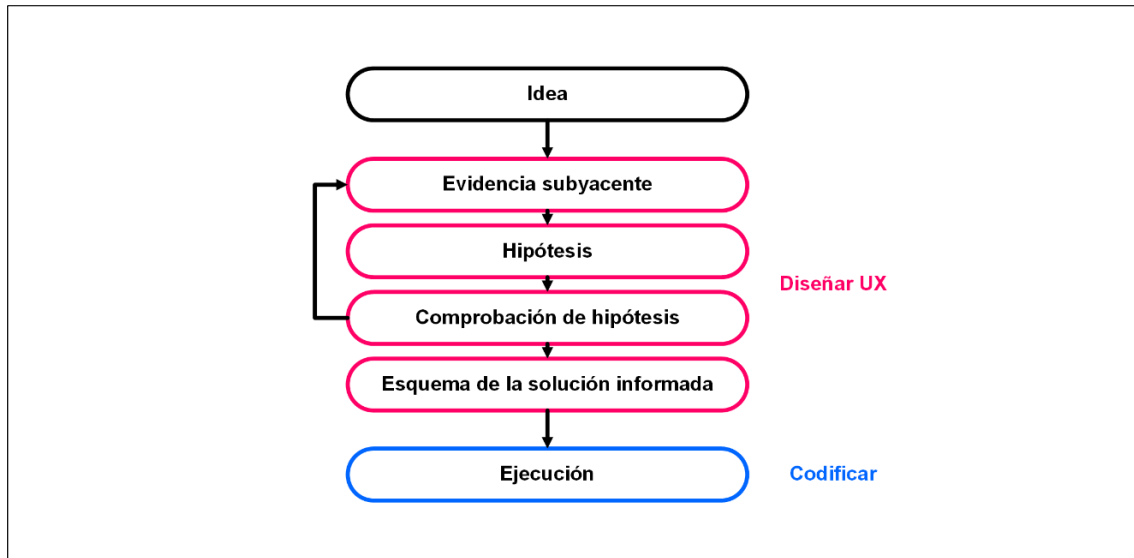
Figura 29. **Desarrollo de software sin proceso de diseño de UX**



Fuente: elaboración propia, realizado con Microsoft Visio.

Los diseñadores de UX siempre necesitan cuestionar las cosas para extraer algún tipo de información, mapearlas y luego sacar las conclusiones junto con los compañeros de equipo. Ideando y creando prototipos de las soluciones y luego probar y comprender si eso funciona o no, o en qué dirección se debe pivotar o actualizar las características.

Figura 30. **Desarrollo de software con proceso de diseño de UX**



Fuente: elaboración propia, realizado con Microsoft Visio.

El diseño siempre se ha tratado sobre la funcionalidad en el propósito y la lógica y aunque el buen diseño a menudo es estéticamente agradable, eso no significa que salió de la nada, hay razones detrás de los colores de los botones, los sistemas de cuadrícula, todo eso debe tener sentido para tener un buen producto usable.

4.2. **Colaboración entre diseñadores y desarrolladores**

La colaboración entre el diseñador y el desarrollador en cualquier proyecto es imperativa para asegurarse de que el proyecto tenga un muy buen resultado.

4.2.1. Consejos de colaboración para diseñadores

El consejo de colaboración que cubre a todos los demás consejos y hace que los demás consejos realmente funcionen es pensar en los demás como más importantes que usted.

Si se piensa en los demás como más importantes que usted mismo, si se piensa que el tiempo de los demás es más importante que su tiempo, entonces realmente se hará todo lo posible para que el proceso sea fluido, ayudará a ser efectivo y eficiente en lo que se hace y realmente la entrega de trabajo de un individuo a otro será mucho más fácil.

4.2.1.1. Incluir a los desarrolladores de principio a fin

Incluir a los desarrolladores de principio a fin, preguntarles qué limitaciones podría tener el proyecto, qué piensan sobre esto, qué opinan sobre la solución que se está planeando. Porque son desarrolladores, son inteligentes y con suerte han hecho docenas y docenas de otros proyectos, ya sean aplicaciones o sitios web, y podrían tener algunas soluciones e ideas realmente buenas.

No se debería creer en la afirmación de que todo el mundo es diseñador, pero sí en la declaración de que todo el mundo tiene técnicas creativas para resolver problemas.

Se deben mostrar las herramientas que se tienen; jugar un poco con ellas e incluir a los desarrolladores desde el comienzo hasta el final del proyecto. De esta manera para cuando se les entreguen las cosas, ellos serán conscientes de

lo que se está consiguiendo, conocerán el compromiso con el proyecto y tendrán algo de aceptación en todo lo que van a construir.

4.2.1.2. Documentar los diseños

Documentar los diseños, tomando en cuenta que actualmente ya no es suficiente entregar a la gente un diseño plano. Si solo se documenta un diseño plano surgirán muchas preguntas por parte de quien recibe el diseño. Así que se deben documentar las fuentes utilizadas, espaciado, interlineado, sistemas y módulos que existen dentro del diseño. Además, se deberían diseñar las interacciones y animaciones. Todo esto para que quien reciba el diseño pueda saber exactamente como se ve y cómo funciona cada parte del diseño.

Documentar el diseño puede hacerse de muchas maneras diferentes, algunas personas usan el tablero de Trello, algunas personas usan *Pivotal Tracker*, algunas personas solo usan *Notes*, algunas personas usan diferentes *plugins* (complementos) para *Sketch* como *Sketch Measure* o *InVision* o cualquier otro, pero se debe tener algo que le diga más al desarrollador que solo la sugerencia de diseño.

4.2.1.3. Crear prototipos

Crear prototipos en lugar de diseños estáticos. Esto no siempre es posible debido a las restricciones de tiempo y otras cosas que pueden suceder, pero mostrar a alguien cómo algo debe verse y sentirse con una animación o una interacción es bastante mejor que solo mostrar un diseño estático.

Para animaciones simples de pantalla a pantalla se puede utilizar algo como *Adobe Experience Designer* o *InVision*, pero para animaciones complejas y con más detalle se puede utilizar algo como *Framer* o *Flinto* o *Principle*.

4.2.1.4. Ser muy organizado

Ser muy organizado. Al entregar cosas a un desarrollador asegurarse que todo este perfectamente organizado para que sea fácil de trabajar.

Asegurarse de que todo este nombrado de cierta manera y todo este ubicado en un determinado directorio y que el archivo sea realmente perfecto a nivel de píxeles.

Establecer sistemas de diseño a lo largo del diseño e implementarlos en todas partes para que el punto sea fácil de entender. Ya que no se sabe si es la primera, la segunda o la tercera vez que alguien mira el proyecto, por lo que se necesita consistencia, se necesita organización, se necesita que todo este realmente bien estructurado para cuando llegue a los desarrolladores.

No debería haber preguntas, todo debería estar realmente muy organizado. Organizar las notas, archivos, diseños, correos electrónicos y así los desarrolladores recibirán cualquier archivo o sugerencia de una mejor manera y el desempeño de ambas partes en el proyecto será mucho mejor.

4.2.1.5. Adaptar sus procesos

Adaptar sus procesos. Al entrar en el flujo de trabajo de alguien más, no dictar qué herramientas usar y cómo diseñar y cómo hacer que las cosas

sucedan, usted tiene su manera de hacer las cosas y ellos tienen su manera de hacer las cosas.

Adaptarse un poco a los procesos y sistemas de alguien más, para dar ese primer paso y mostrar que se está dispuesto a comprometerse. Eso creará un puente entre los dos. Así que se debe estar dispuesto a adaptarse un poco, según sea necesario, no revelar todo su proceso ni sacrificarlo todo, pero probablemente haya algunas cosas que se puedan hacer para que los desarrolladores se sientan más cómodos con lo que se les va a entregar.

4.3. Colaboración entre diseñadores de UX y *product managers*

Los diseñadores de UX son responsables de comprender los desafíos del cliente al usar un producto en particular y luego diseñar una solución elegante que ayude a los usuarios a resolver esos desafíos. Sucede que el objetivo de proporcionar una solución a los clientes se superpone con los objetivos del *product manager*. Por lo que existe una confusión en cuanto a dónde termina un rol y el otro comienza.

De hecho, en muchas *startups* pequeñas, el *product manager* generalmente termina asumiendo el papel del diseñador de UX, al hacerlo, se espera que comprenda el mercado de productos que su empresa tiene, objetivos comerciales que el usuario necesita y la mejor manera de priorizar las características y administrar un equipo. Pero a medida que los equipos se hacen más grandes y los *stakeholders* se vuelven más numerosos, los dos roles se dividen en posiciones más especializadas.

Tanto el diseñador de UX como el *product manager* comparten el mismo objetivo final, que es crear el mejor producto posible con un increíble diseño de Experiencia de Usuario.

4.3.1. *Product manager* (gerente de producto)

Principalmente los *product managers* son responsables del éxito general del producto, los *product managers* son muy parecidos a los CEOs.

Principales tareas de un *product manager*:

- Establecer objetivos para encontrar el éxito
- Ayudar a motivar a los equipos
- Coordinar con diferentes departamentos y *stakeholders*
- Equilibrar las necesidades de tecnología, UX y *stakeholder* para administrar las compensaciones que surgen.
- Definir los requisitos y priorizar las características, determinando así lo que debe hacer el equipo de desarrollo de productos, esto implica tareas como elegir de la larga lista de ideas lo que es importante, realista y urgente en el ciclo de vida de un producto y actuar sobre ello.

Los *product managers* son el punto de referencia en un equipo cuando se trata de comprender las necesidades del usuario y los fundamentos del diseño de un producto.

Los buenos *product managers* siempre trabajan con base en un *framework*, estos proporcionan los pasos básicos involucrados en el *roadmap* del producto de una empresa desde la idea hasta el lanzamiento.

Los pasos que las empresas utilizan para el desarrollo del producto varían en función de la estrategia del producto que se está implementando, actualmente y por lo general se utilizan hasta 11 pasos:

- *Ideation* (ideación)
- *Research* (investigación)
- *Design* (diseño)
- *Prototype* (prototipo)
- *Development* (desarrollo)
- *Document* (documentación)
- *Test* (prueba)
- *Train* (entrenamiento)
- *Release* (lanzamiento o liberación)
- *Maintenance* (mantenimiento)
- *Retirement* (retiro) (solo en última instancia)

4.3.2. Diseñador de UX

Responsabilidades de un diseñador de UX:

- Garantizar la usabilidad del producto

Se pasa por un proceso para optimizar las aplicaciones y crear la mejor Experiencia de Usuario, explorando diferentes enfoques para crear la mejor solución posible para los clientes.

- Realizar estudios e investigaciones de usabilidad

Esto para soportar la solución que se crea.

Estudiar el comportamiento de los usuarios para así poder refinar y modificar el diseño basándose en las necesidades encontradas.

- Definir el diseño del producto y su funcionamiento

Evaluar la Experiencia de Usuario con productos con los que el usuario ya trabaja.

- Comprender las necesidades de los usuarios

4.3.3. *Product manager* y diseñador de UX

Para que los *product managers* trabajen de manera efectiva junto con los diseñadores de UX, es importante que:

- Los *product managers* comprendan las dimensiones de su función.

Los *product managers* entiendan que son responsables del panorama general.

Es decir, colaborar entre todos los grupos de productos haciendo un caso sólido para la visión y definición del producto.

- Los *product managers* fomenten una relación profesional interdependiente con los diseñadores de UX.

Los *product managers* deben trabajar en estrecha colaboración con los diseñadores de UX.

Para comprender mejor a los usuarios de productos porque los diseñadores de UX son en virtud de su papel, generalmente los más cercanos a los usuarios reales.

Los *product managers* deben buscar a los diseñadores de UX por su experiencia, diseño de productos y estudios de usabilidad, mientras que al mismo tiempo prestan un ojo crítico y se aseguran de que todo tenga sentido en el contexto más amplio.

Los diseñadores de UX a su vez también se benefician de una buena comprensión del contexto empresarial y de reconocer que hay muchos otros factores involucrados que un *product manager* debe tener en cuenta.

4.4. Relación entre DevOps y diseño de UX

DevOps es un conjunto de prácticas de desarrollo de software que combina el desarrollo de software y las operaciones de tecnología de la información para acortar el ciclo de vida de desarrollo de los sistemas, al tiempo que se entregan características, correcciones y actualizaciones con frecuencia en estrecha alineación con los objetivos empresariales. (DeBoer, 2019)

El diseño de la experiencia del usuario es el proceso de aumentar la satisfacción del usuario con un producto mediante la mejora de la usabilidad, la accesibilidad y la conveniencia en la interacción con un producto, al tiempo que se alinea con los objetivos empresariales. (Minhas, 2018)

Gran parte de DevOps es entregar características, correcciones y actualizaciones, mientras que estas entregas están alineadas con los objetivos empresariales, están orientadas a resolver problemas para un usuario final. Gran parte del diseño de UX es entregar correcciones y actualizaciones de características que mejoren la experiencia de un usuario con el producto.

4.4.1. Habilidades necesarias para ser un ingeniero de DevOps y para ser un diseñador de UX

En la siguiente tabla se presentan las habilidades necesarias para ser un ingeniero de DevOps y para ser un diseñador de UX.

Tabla III. **Habilidades necesarias para ser un ingeniero de DevOps y para ser un diseñador de UX**

Habilidades necesarias para ser un ingeniero de DevOps	Habilidades necesarias para ser un diseñador de UX
Duma (2018)	
Gran capacidad de comunicación y colaboración.	Gran capacidad de comunicación y colaboración.
Empatía y desinterés.	Empatía y desinterés.
Conocimiento de las principales herramientas DevOps.	Conocimiento de las principales herramientas de diseño.
Habilidades de seguridad de software.	Habilidades de accesibilidad.
Dominio de las tecnologías y herramientas de automatización.	Dominio de las herramientas de creación de prototipos y pruebas.
Habilidades de codificación y <i>scripting</i> (secuencia de comandos).	Habilidades de diseño y <i>user research</i> (investigación de usuarios).
Habilidades en la nube.	Habilidades en la nube.
Habilidades en <i>testing</i> (pruebas).	Habilidades en <i>user testing</i> (pruebas de usuario).
Mentalidad centrada en el cliente.	Mentalidad centrada en el usuario.
Pasión y proactividad.	Pasión y proactividad.
(párrafos varios)	

Fuente: elaboración propia.

Existen similitudes bastante claras entre las habilidades necesarias para ser un ingeniero de DevOps y las habilidades necesarias para ser un diseñador de UX, son muy cercanas, hay algunas diferencias en las tecnologías, pero hay mucha superposición.

4.4.2. Principios de DevOps y diseño de UX

Tanto en DevOps como en diseño de UX existen varios principios Ágiles que se siguen para trabajar. Si bien es cierto que no existe un estándar y cada

autor o empresa establece sus principios Ágiles con base en sus conocimientos, experiencias y según se adapte a su situación particular. Ya sea que los principios sean establecidos para DevOps o para diseño de UX aplican de manera correcta en ambos casos.

New Relic (s.f.) Ejemplo de principios Ágiles para DevOps:

- Establecer las expectativas, las prioridades y las creencias fundamentales que los guían.
- Colaborar tanto dentro de los equipos como entre ellos en la resolución de problemas.
- Automatizar los procesos comunes y repetitivos para liberar tiempo para el trabajo de mayor nivel.
- Integrar la retroalimentación en el trabajo, midiendo todo lo que se pasa a producción.
- Compartir los datos con todos los implicados para fomentar una cultura más eficaz de trabajo conjunto entre diferentes habilidades y conocimientos especializados.

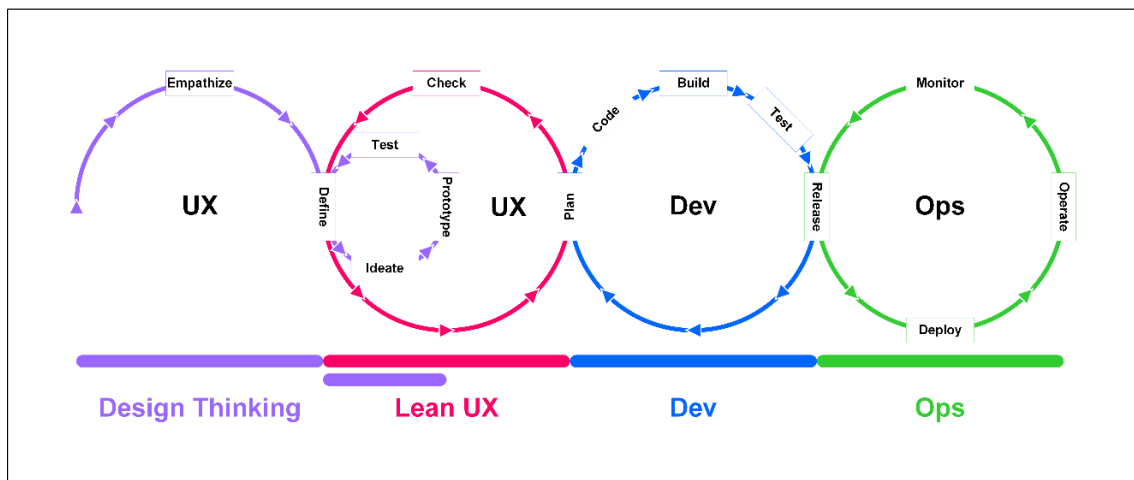
5. INTEGRACIÓN DEL PROCESO DE DISEÑO DE *USER EXPERIENCE* AL CICLO DEVOPS CI/CD

5.1. UXDevOps

La integración de UX con DevOps es esencial para que los equipos de DevOps se aseguren de resolver los problemas correctos y comprendan las necesidades reales del usuario.

UXDevOps es un ciclo de desarrollo de software centrado en el usuario, capaz de crear soluciones, iterar en ellas y validarlas a través de la experimentación, entregando mejoras incrementales.

Figura 31. UXDevOps



Fuente: elaboración propia, realizado con Microsoft Visio.

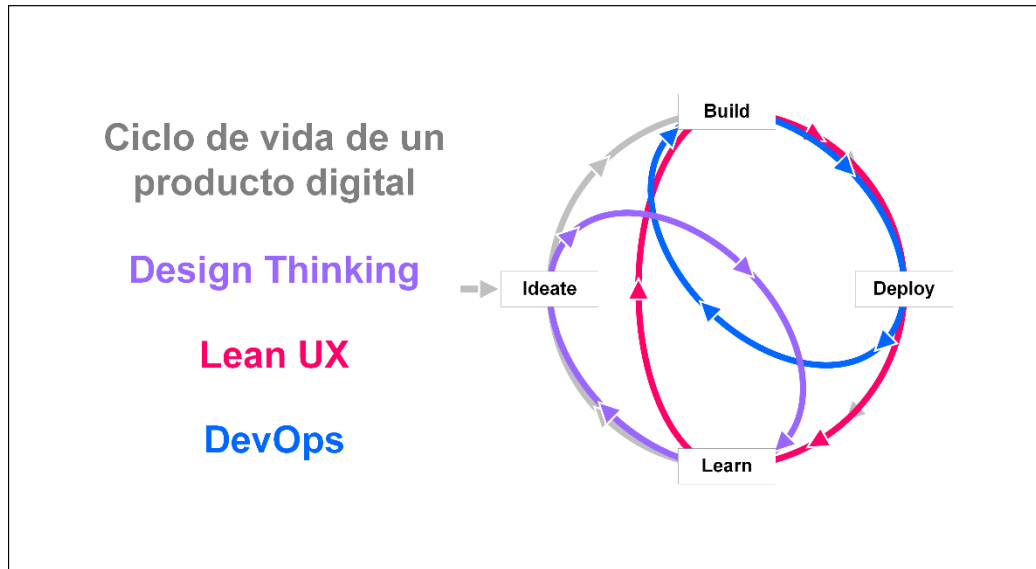
Surge al combinar tres ciclos, dos ciclos de diseño de UX y un ciclo de desarrollo de software. *Design Thinking*, *Lean UX* y DevOps CI/CD respectivamente.

El ciclo comienza con la fase *empathize* de *Design Thinking*. Luego entra al ciclo de *Lean UX*, donde la fase *think* es un ciclo con varias fases de *Design Thinking*, la fase *make* donde inicia el ciclo DevOps CI/CD y la fase *check*.

El proceso de diseño de UX formado por *Design Thinking* y *Lean UX* es realmente genial, lleva por un camino en el que se realiza un trabajo de diseño y validación de ideas, para luego saber exactamente qué tipo de características o ideas se van a impulsar en el Ciclo DevOps CI/CD que se trata de ejecución.

Esta es una forma muy efectiva de avanzar con un proyecto teniendo bajo riesgo. Al combinar estos tres esencialmente se tiene una parte manejable de trabajos, fragmentos realistas de trabajos, que se han validado y se avanza con una forma muy eficiente de producción.

Figura 32. Partes UXDevOps



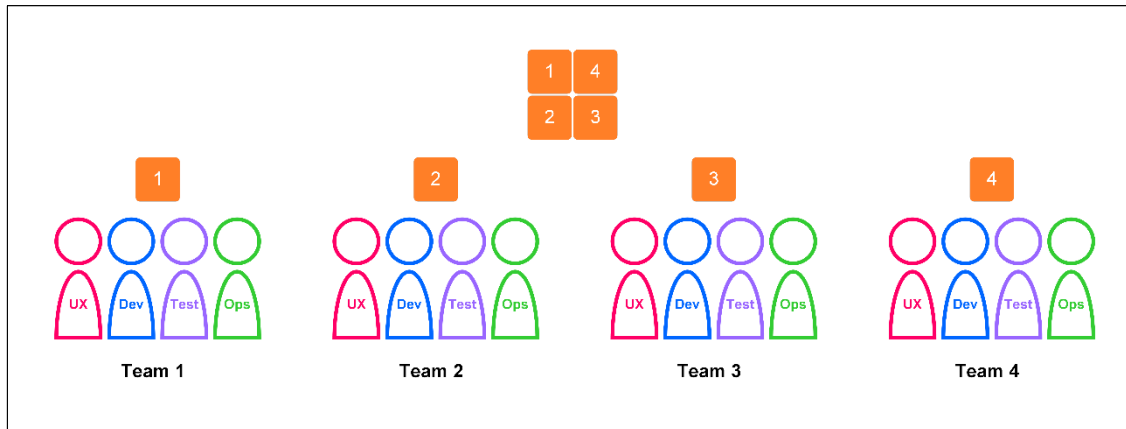
Fuente: elaboración propia, realizado con Microsoft Visio.

5.1.1. Equipos en UXDevOps

La observación conocida como la Ley de Conway establece que la forma en que se organizan los equipos afecta el software que construyen, incluida su arquitectura, se debe reorganizar la forma en que trabajan las personas, se debe asegurar que el código de alta calidad se implemente rápidamente y hacer que la Ley de Conway funcione a favor del software.

Los equipos en UXDevOps consisten en pequeños equipos multifuncionales que trabajan de forma independiente en un determinado sistema, parte o servicio.

Figura 33. Equipos en UXDevOps



Fuente: elaboración propia, realizado con Microsoft Visio.

No existe un número exacto de la cantidad de personas que deben conformar un equipo pequeño, sin embargo, se considera que debería ser igual o alrededor de cinco personas, un par más o un par menos. La idea principal aquí es fusionar todas las habilidades y tareas en un solo equipo de servicio, incluido el diseño de UX, la ingeniería de software, garantía de calidad, seguridad y operaciones.

La actitud tiene sentido a medida que los miembros del equipo trabajan hacia un objetivo común, un software intuitivo, fácil de usar, estable, seguro y confiable.

Un equipo en las organizaciones UXDevOps admite el ciclo de vida completo del desarrollo de software, desde el diseño de UX y la escritura del código hasta el mantenimiento del producto después del lanzamiento.

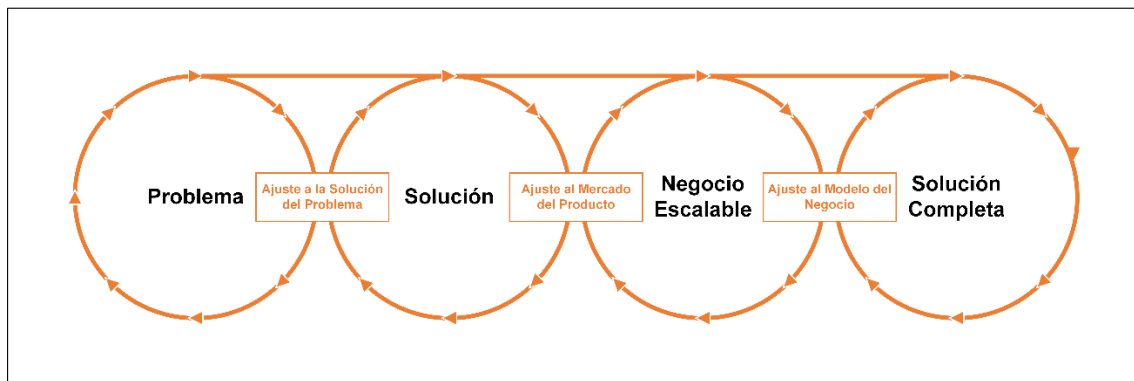
5.1.2. Proceso en UXDevOps

Se identifican tres tipos de ajuste en el proceso:

- Ajuste a la solución del problema. Para lograrlo se debe saber cuál es exactamente el problema que se necesita resolver.
- Ajuste al mercado del producto. Para lograrlo se necesita entender cuál sería la mejor solución.
- Ajuste al modelo de negocio. Para lograrlo se necesita entender cómo hacer un negocio escalable a partir del ajuste al mercado del producto.

Finalmente, también se debe poder ofrecer una solución completa.

Figura 34. Ajustes Proceso UXDevOps



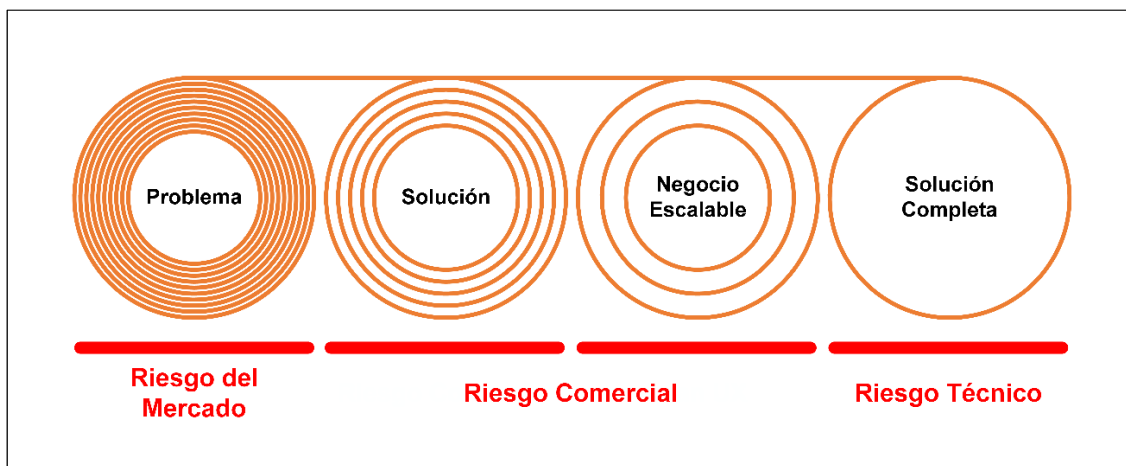
Fuente: elaboración propia, realizado con Microsoft Visio.

Es posible identificar los distintos riesgos a lo largo del proceso. Al inicio con el problema se enfrenta el riesgo del mercado, luego cuando se busca la

solución correcta y el negocio escalable se enfrenta el riesgo comercial y por último con la solución completa se enfrenta el riesgo técnico.

Esto también está en línea con la estructura del proceso, al principio es muy caótico porque existen muchas incógnitas, más adelante se vuelve un poco más estable y finalmente se sabe qué es exactamente lo que se quiere resolver.

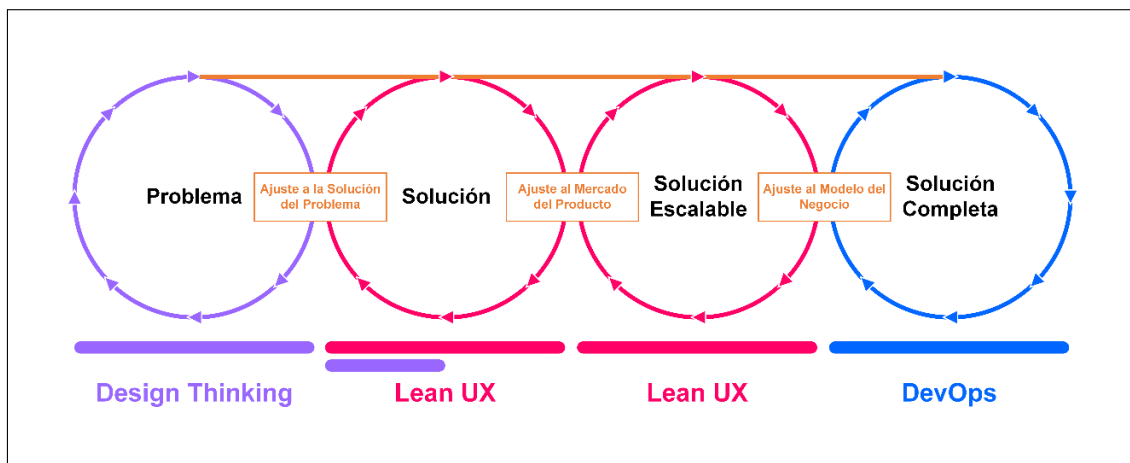
Figura 35. **Riesgos Proceso UXDevOps**



Fuente: elaboración propia, realizado con Microsoft Visio.

Cada una de estas diferentes etapas requiere un proceso para usar. Al principio se usa *Design Thinking*, luego se usa *Lean Startup* y por último se usa DevOps.

Figura 36. **Proceso UXDevOps**



Fuente: elaboración propia, realizado con Microsoft Visio.

- *Design Thinking*. En esta etapa, no está claro dónde está exactamente el mercado, quiénes son los diferentes *stakeholders* y cuál es el problema que se está tratando de resolver. Hay una alta incertidumbre, una alta ambigüedad y es muy caótico.
- *Lean UX*. En esta etapa se logra que el producto se ajuste al mercado y se encuentre el modelo de negocio.

Es muy adecuada para construir el negocio y encontrar la mejor solución para los clientes objetivo y un modelo de negocio escalable, es por eso por lo que se vincula al ajuste al mercado del producto y al ajuste al modelo del negocio.

- DevOps. Esta etapa se trata sobre desarrollar la solución

Este proceso de desarrollo está en línea con el proceso de *Lean UX*, ya que *Lean UX* necesita hacer pequeños cambios con bastante frecuencia y el proceso de desarrollo con DevOps es capaz de hacer esto.

CONCLUSIONES

1. El Ciclo DevOps CI/CD en UXDevOps se encarga de construir y publicar. Es un flujo de trabajo veloz y eficiente, capaz de adaptarse a las necesidades y entregas frecuentes de los procesos de diseño de UX, entregando mejoras incrementales que son soluciones al problema que se está resolviendo y que fueron previamente diseñadas en los procesos de diseño de UX.
2. El proceso de diseño de UX en UXDevOps se encarga de diseñar y validar. Inicia el proceso reuniendo información y descubriendo qué problema se está resolviendo, luego se diseñan soluciones que más adelante serán construidas y publicadas en el Ciclo DevOps CI/CD, después las soluciones se validan y se repite el ciclo.
3. El diseño de UX en su definición más simple es diseñar software y el Ciclo DevOps CI/CD en su definición más simple es desarrollar software. Por lo tanto, si se combinan se tiene un ciclo para el desarrollo software centrado en el usuario, que reduce el tiempo y el esfuerzo, y produce software que realmente resuelve las necesidades del usuario.
4. UXDevOps es un ciclo de desarrollo de software que se centra en el usuario, combina varios ciclos de diseño de UX y el Ciclo DevOps CI/CD. Toma en cuenta todos los pasos del proceso de desarrollo de un producto digital, idear, construir, desplegar y aprender.

RECOMENDACIONES

1. Adoptar una mentalidad y cultura DevOps. Mas allá de lo que son las prácticas y tecnologías de DevOps, se necesita un cambio de mentalidad y cultura entre los miembros del equipo, para que la colaboración entre los involucrados funcione de la manera correcta.
2. Dar la debida importancia al diseño de UX en el desarrollo de software. Muchas veces en el desarrollo de software el diseño de UX suele ser excluido, porque se piensa que no tiene importancia, cuando la realidad es que es uno de los procesos que mayor importancia tiene, ya que permite saber si lo que se desarrollara es realmente lo que el usuario necesita, antes de desperdiciar tiempo y recursos en su desarrollo.
3. Comprender como funcionan, como se complementan y los puntos en común de los procesos del diseño de UX y del Ciclo DevOps CI/CD. De esta manera se conoce todo el sistema desde distintas perspectivas y se puede ser empático y trabajar de una forma eficiente con otros equipos y miembros del equipo.
4. Adaptar UXDevOps a los proyectos donde se quiera usar. UXDevOps es bastante flexible, no se pretende que se siga como una receta paso a paso, sino que sirva como una herramienta para dar una dirección clara.

REFERENCIAS

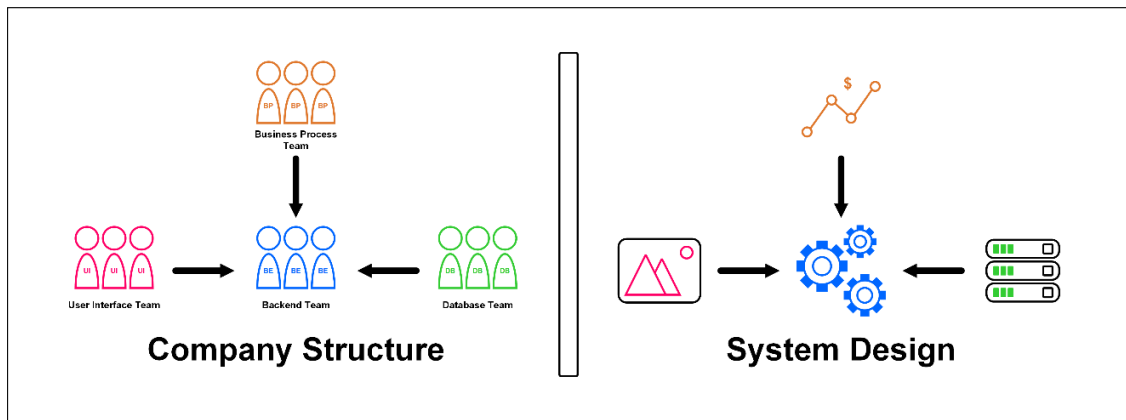
1. Agile Manifesto (s.f.) *Manifiesto por el Desarrollo Ágil de Desarrollo*. [Mensaje en un blog]. Recuperado de <https://agilemanifesto.org/iso/es/manifesto.html>.
2. Agil Manifesto. (6 de marzo de 2022). *Principios del Manifiesto Ágil*. [Mensaje en un blog]. Recuperado de <https://agilemanifesto.org/iso/es/principles.html>.
3. Brown, T. *Change by Design*. (1a edición). Estados Unidos: HarperCollins Publishers Inc.
4. Courtney, J. (27 de marzo de 2022). *The Design Sprint 2.0: What is it and what does it look like?*. [Mensaje en un blog]. Recuperado de <https://www.invisionapp.com/inside-design/design-sprint-2/>.
5. Deboer, E. (27 de marzo de 2022). *What is the Definition of DevOps?*. [Mensaje en un blog]. Recuperado de <https://blog.sonatype.com/definition-of-devops>.
6. Dhaduk, H. (13 de febrero de 2020). *DevOps Lifecycle: 7 Phases Explained in Detail with Examples*. [Mensaje en un blog]. Recuperado de <https://www.simform.com/blog/devops-lifecycle/>.

7. Duma, Y. (12 de diciembre de 2018). *Ten Skills Every DevOps Engineer Must Have for Success*. [Mensaje en un blog]. Recuperado de <https://squadex.com/insights/devops-skills-success/>.
8. Gilson, N. (28 de diciembre de 2021). *What is Conway's Law?* [Mensaje en un blog]. <https://www.atlassian.com/blog/teamwork/what-is-conways-law-acmi>.
9. Google Cloud. (2021). *Accelerate State of DevOps 2021*. Estados Unidos: Autor.
10. Halvorson, K. (16 de diciembre de 2008). *The Discipline of Content Strategy*. [Mensaje en un blog]. Recuperado de <https://alistapart.com/article/thedisciplineofcontentstrategy/>.
11. Kanban (2020) *The Kanban Guide*. Estados Unidos: Vacanti, Inc.
12. Knapp, J., Zeratsky J., Kowitz B. (2016). *Sprint*. (1a edición) Estados Unidos: Simon & Schuster, Inc.
13. MINHAS, S. (23 de abril de 2018). *User Experience Design Process*. [Mensaje en un blog]. Recuperado de <https://uxplanet.org/user-experience-design-process-d91df1a45916>.
14. New Relic. *What Is DevOps?*. [en línea]. <<https://newrelic.com/devops/what-is-devops>>. [Consulta: 27 de marzo de 2022].

15. Nielsen, J. (3 de enero de 2012). *Usability 101: Introduction to Usability*. [Mensaje en un blog]. Recuperado de <https://www.nngroup.com/articles/usability-101-introduction-to-usability/>.
16. Project Management Institute. (2021) *Guía del PMBOK*. (7ª Edición) Estados Unidos: Project Management Institute, Inc.
17. Ries, E. (2011) *The lean startup*. (1ª edición). Estados Unidos: RandomHouse, Inc.
18. Schwaber, K., Sutherland J. (2020). *The scrum guide*. (1a edición). Estados Unidos: Schwaber Inc.
19. Wells, D. (8 de octubre de 2013). *Extreme Programming: A gentle introduction*. Agile Process. Recuperado de <http://www.extremeprogramming.org/>.
20. Wickramasinghe, S. (16 de abril de 2021). *What Is DevOps? A Comprehensive Introduction*. [Mensaje en un blog]. Recuperado de <https://www.bmc.com/blogs/devops-basics-introduction/>.

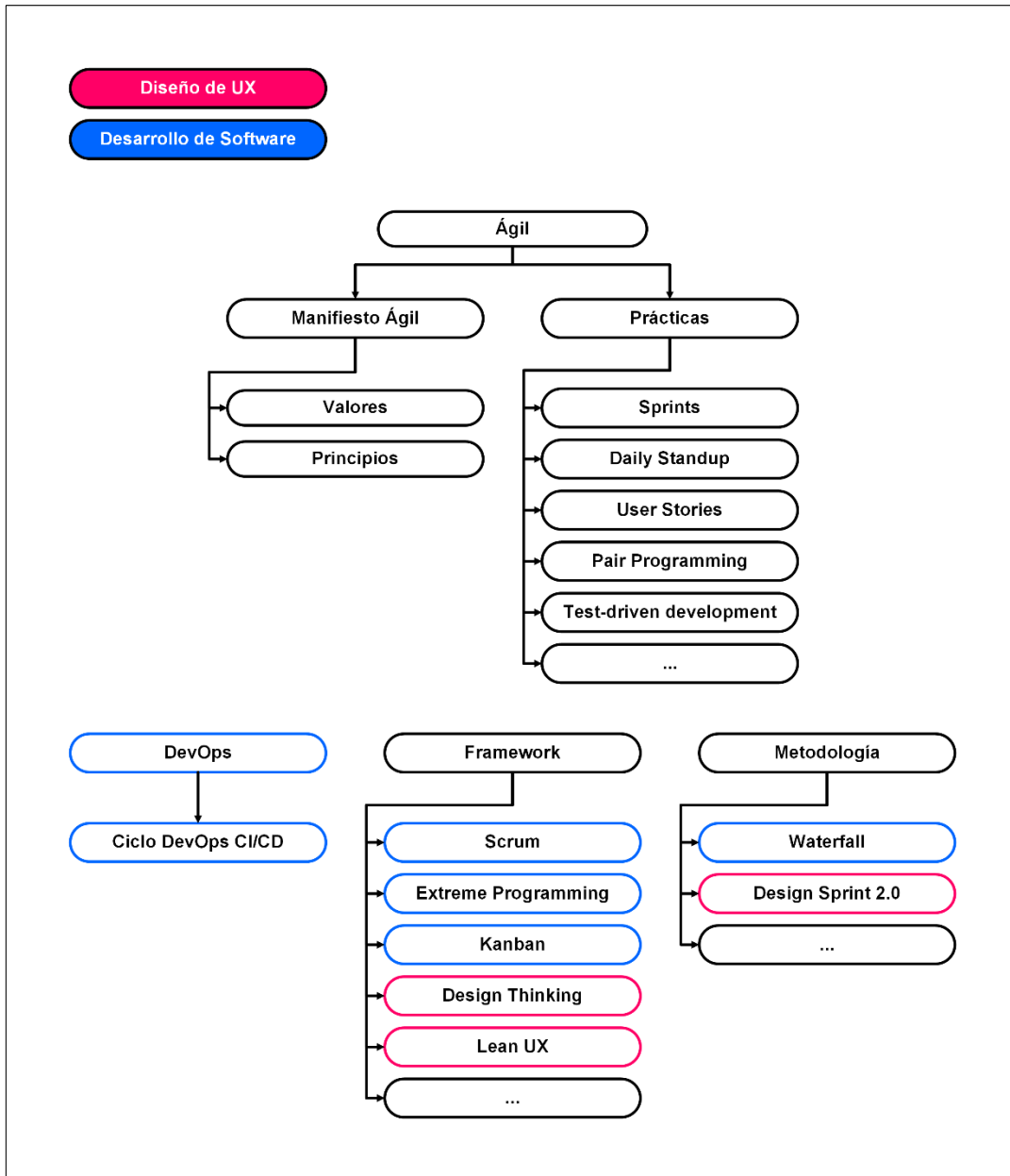
APÉNDICES

Apéndice 1. Ley de Conway



Fuente: elaboración propia, realizado con Microsoft Visio.

Apéndice 2. Diseño de UX y desarrollo de software



Fuente: elaboración propia, realizado con Microsoft Visio.