Actividades por equipo

Integrantes:

-Betancourt Velázquez Nancy Etzel 181080029

-Negrete Quiroz Alejandro 181080153

-Garrido de la Cruz Karla Gabriela 181080157

-Rojas Hernández Axel Joel 181080176

Materia:Fundamentos de Ingeniería de software

Grupo: ISC-5AV SCC 1007

El propósito de la Ingeniería de software es desarrollar soluciones que sean automatizadas en el ámbito real por personas u organizaciones que tengan en común el mismo interés en el cual tengan las herramientas, técnicas y métodos que serán utilizados en el desarrollo de los sistemas de software.

A finales de 1967 Friedrich Ludwic Bauer argumento que era urgente aplicar la ingeniería en el proceso del software ya que ésta atravesaba una crisis, se realizaron conferencias en alemania en el año 1969 y en italia eb 1969 en la cuales se analizaron el diseño, la calidad del software y fue entonces cuando se fueron acumulando métodos, técnicas y buenas prácticas para tener conocimientos maduros para formar profesionistas especializados en el desarrollo del software.
En 2004 se logró compilar el conocimiento y se establecieron subconjuntos de áreas de las cuales 5 se vincularon con los procesos de desarrollo y 6 en las áreas de gestión.

Las áreas de desarrollo se presentan de la siguiente manera, las cuales constan de cuatro faces que se vinculan al mantenimiento de software.

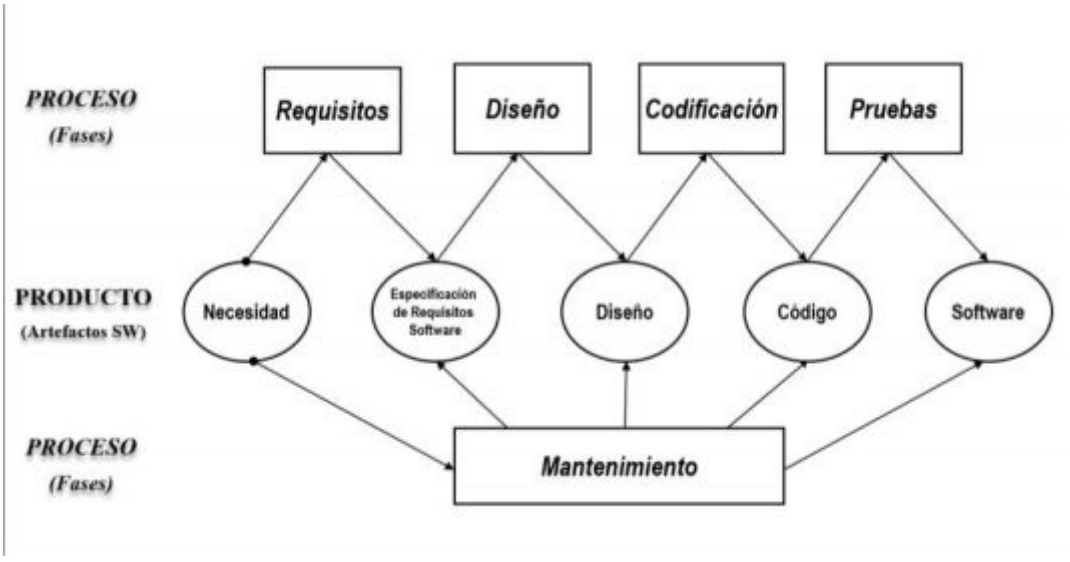


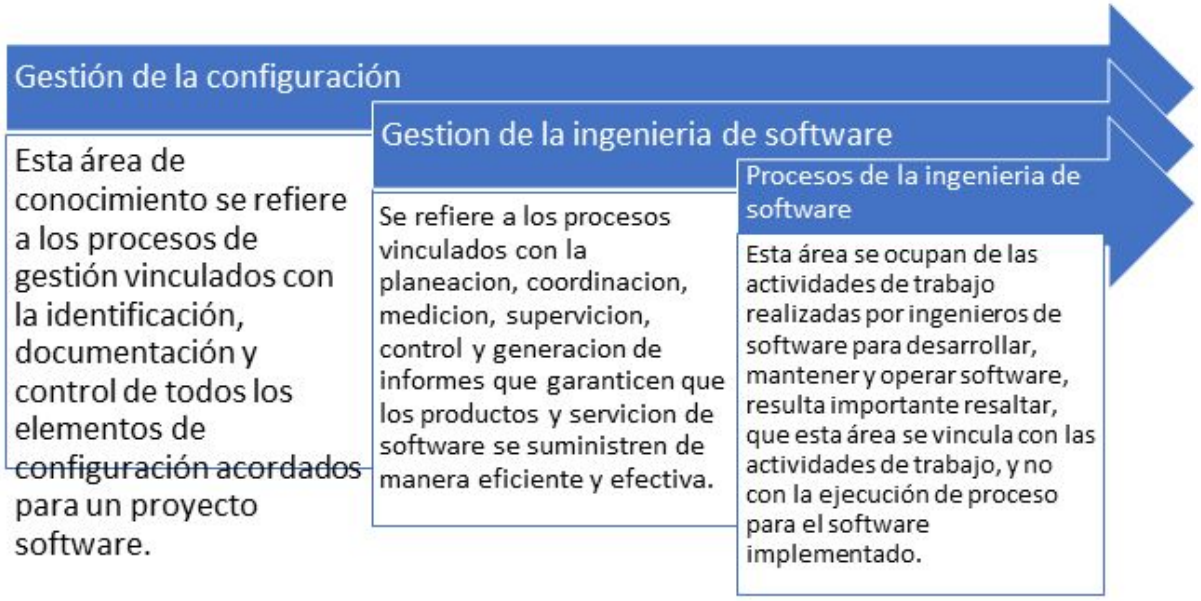
Figura 1.1 Dualidad Proceso-Producto en la Ingeniería de Software

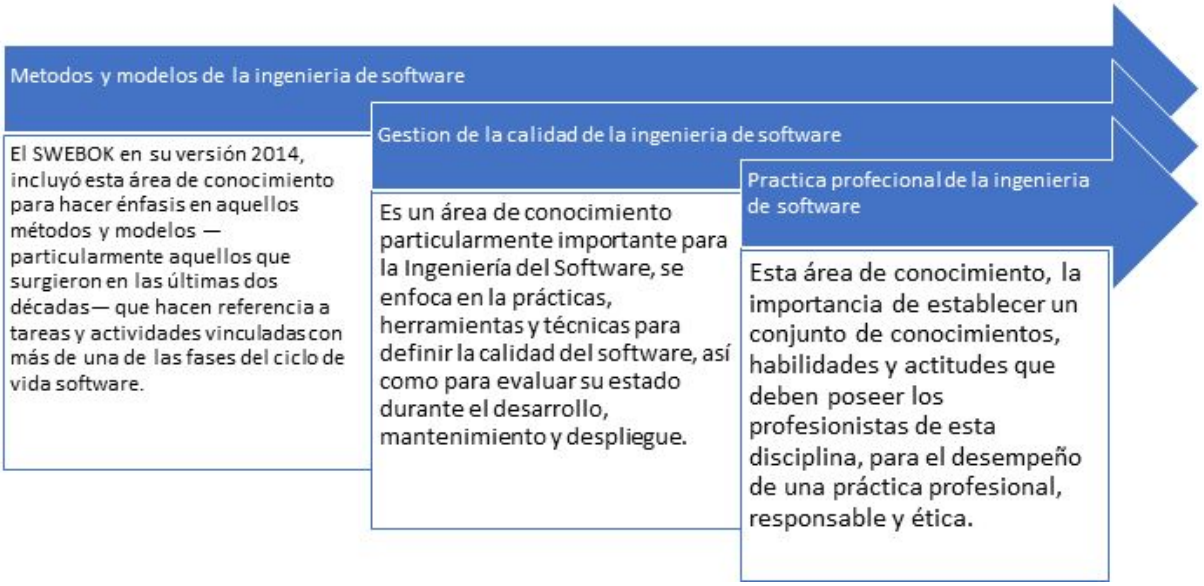
Desarrollo de requisitos: Son las necesidades y restricciones respecto de un producto de software.
Diseño de software: Define tanto la arquitectura, componentes, interfaces, modelos de persistencia de los datos y los resultados.

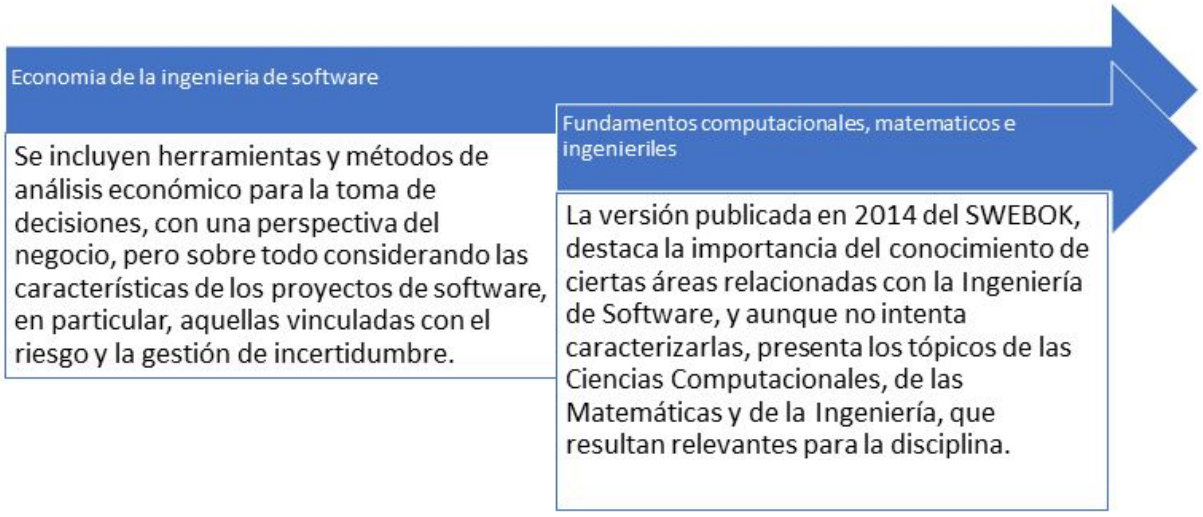
Programación: Es la creación a detalle del software por medio de la codificación en un lenguaje de programación.

Pruebas: Es la verificación del comportamiento que se espera que tenga el software, se realiza la prueba- error para que cumpla todos los requisitos de aceptación de los clientes.

Mantenimiento: Ayuda a que el software tenga las menores fallas posibles, así como la integración de nuevas cosas o actividades, puede ser un mantenimiento correctivo, perfectivo, preventivo y adaptativo.







Retos de la Disciplina

La atención está más enfocada en el software funcional, en la aceptación del usuario. Es conveniente abrir líneas de investigación que atiendan esta vertiente; así mismo, hace falta promover el estudio de la ciencia de la computación y de la Ingeniería de Software desde esta perspectiva.

La Ingeniería de Software ha sido aceptada como una disciplina de carácter práctico. Por ser relativamente nueva, comparada con otras disciplinas de ingeniería, le hace falta el carácter de formalización basada en fundamentos teóricos de las matemáticas y la ciencia de la computación. Estas observaciones están relacionadas con la falta de rigurosidad formal utilizada en esta área, y sugieren que esta deficiencia hace ver a la Ingeniería de Software como un arte y no como una disciplina formal.

Los problemas típicos en Ingeniería de Software, son:

(1)Las especificaciones de los requisitos son frecuentemente incompletas, ambiguas, inconsistentes y/o inconmensurables
(2) Los proyectos a menudo se retrasan y exceden el presupuesto
(3) No existen métodos para garantizar que el software entregado "funcionará" para el usuario
(4) Falta de detección sistemática de defectos de software
(5) Los procedimientos, métodos y técnicas para diseñar un sistema de control de proyectos que permitirán a los gerentes de proyectos controlar con éxito su proyecto no están disponibles
(6) Relación poco predecible de la duración del proyecto
Las técnicas de Inteligencia Artificial tienen la capacidad de mejorar el desarrollo de software,es conveniente aumentar el uso de las técnicas de IA para resolver problemas de la Ingeniería de Software. Hoy en día, está siendo más aceptado en este contexto, existen pocos intentos de aplicar técnicas inteligentes en la solución de problemas de la Ingeniería de Software. La diversidad y complejidad computacional de los sistemas que se desarrollarán en los próximos años, implicarán un impacto muy fuerte sobre la Ingeniería de Software que tendrá que afrontar los nuevos problemas que se avecinan. Este análisis presentado por Boehm (2006) nos permite visualizar los campos a los que debe turnarse la Ingeniería de Software, ya que tradicionalmente se ha enfocado más en temas específicos de la misma disciplina y de la Ciencia de la Computación.

Computación en la nube: Basada en Internet que proporciona recursos de procesamiento compartidos y datos, para computadoras y otros dispositivos, en base a demanda.

Computación social: Se refiere a los sistemas que permiten la obtención, representación, procesamiento, uso y difusión de la información que se distribuye a través de colectividades sociales tales como equipos, comunidades, organizaciones y mercados electrónicos.

Datos masivos: Se refiere a conjuntos de datos que son muy grandes y complejos, tanto que las aplicaciones tradicionales de procesamiento son inadecuadas para manejarlos.

Primero, comprendamos el concepto de ingeniería de software.

El término consta de dos palabras: software e ingeniería.

El software es más que un simple código de programa. Los programas son códigos ejecutables que se utilizan con fines computacionales. El software se considera una colección de códigos de programas ejecutables relacionados con bibliotecas y documentación. El software que cumple con requisitos específicos se denomina producto de software.

La ingeniería, por otro lado, intenta desarrollar productos utilizando métodos y principios científicos bien definidos.

La ingeniería de software es una rama de la ingeniería relacionada con el desarrollo de productos de software utilizando métodos, principios y procedimientos científicos.

El resultado de la ingeniería de software es un producto de software eficiente y confiable

Definiciones.

IEEE (Instituto de Ingeniería Eléctrica y Electrónica) define la ingeniería de software como:
(1) La aplicación de métodos sistemáticos, estandarizados y cuantificables en el desarrollo, operación y mantenimiento de software; esta es básicamente la aplicación de la ingeniería en software.
(2) Investigación aproximada como se describe arriba.

El teórico informático alemán Fritz Bauer definió la ingeniería de software como:
La ingeniería de software es el establecimiento y uso de principios de ingeniería sólidos para obtener software confiable y eficiente en máquinas reales de una manera económica.

Paradigmas de software.

El paradigma del software son los métodos y pasos que se realizan al diseñar el software. Se han propuesto muchos métodos y se pueden usar en la actualidad, pero debemos observar la posición de estos ejemplos en el marco de la ingeniería de software.

Estos se pueden combinar en varias categorías, cada una de las cuales contiene otra categoría:

Desarrollo de software:

Este paradigma se denomina paradigma de la ingeniería de software, que implementa todos los conceptos de ingeniería relacionados con el desarrollo de software. Incluye varias encuestas y recopilación de requisitos, lo que ayuda a crear productos de software.

- Cobro por demanda
- Diseño de software
- Programación

Diseño de software.

Este ejemplo es parte del desarrollo de software e incluye:

- Diseño
- Mantenimiento
- Programación

Programación.

Este paradigma está estrechamente relacionado con el aspecto de programación del desarrollo de software.

- Codificación
- Prueba
- Integral

Necesidad de ingeniería de software.

La demanda de ingeniería de software proviene de los rápidos cambios en los requisitos de software y el entorno de trabajo.

Software a gran escala:
De manera similar, a medida que aumenta el tamaño del software, es mucho más fácil construir muros que construir casas, por lo que se debe realizar un diseño de ingeniería para que sea cientifico.

Escalabilidad:
Si el proceso de software no se basa en conceptos científicos y de ingeniería, es más fácil recrear software nuevo que ampliar el software existente.

Costo:
A medida que la industria del hardware ha demostrado sus capacidades y sus excepcionales capacidades de fabricación, el precio del hardware electrónico y de computadora ha caído. Sin embargo, si el proceso no puede adaptarse a los nuevos desarrollos, el costo del software sigue siendo alto.

Naturaleza dinámica:
La naturaleza del software crece y se adapta constantemente, en gran medida dependiendo del entorno de trabajo del usuario. Si la naturaleza del software cambia constantemente, será necesario mejorar el software existente. Aquí es donde la ingeniería de software juega un papel importante.

Gestión de calidad:
El mejor proceso de desarrollo de software puede producir productos de mejor calidad.

TECNOLÓGICO NACIONAL DE MÉXICO®

Introduccion a la ingenieria de Software

Ingenieria software: su propósito es el desarollar soluciones automatizadas a necesidades reales expresadas por personas u organizaciones con intereses en común.

se conforma de un conjunto de :

1. técnicas
2. herramientas
3. métodos
4. procesos

utilizados para el proceso de desarrollo , operación y mantenimiento

El desarrollo del software se describe en cinco fases:

1. requisito de diseño
2. codificación
3. pruebas
4. mantenimiento

Areas de desarrollo	
Desarrollo de requisitos	conjunto de necesidades y restricciones expresadas respecto de un producto software
Diseño de software	definicion de la arquitectura , componentes, interfaces, modelos de persistencia de datos y el resultado del mismo
programacion o fase de programacion o codificacion	creacion detallada del software, se hace uso de un lenguaje de programacion
Pruebas	conjunto de procesos vinculados del comportamiento esperado del software
Mantenimiento	tiene como proposito modificar el software existente y preservar su identidad: correctivo, perfectivo, preventivo y adaptativo

Areas de Gestion	
Gestion de la configuracion	contiene caracteristicas volatiles, procesos de gestion: identificacion, documentacion y control
Gestion de ingenieria de software	procesos vinculados con la planeacion, coordinacion, medicion, supervicion, control y generacion de informes
Procesos de ingenieria de software o procesos de software	actividades de trabajo realizadas por ing. Para desarrollar , mantener y operar software
Metodos y modelos de ingenieria de software	Tareas y actividades vinculadas con mas de una de las fases del ciclo de vida software
Gestion de calidad del software	practicas, herramientas y tecnicas para definir la calidad del software, evaluar su desarrollo , mantenimiento y despliegue
Practica profesional de la ingenieria de software	establece un conjunto de conocimientos, habilidades, y actitudes
Economia de la ingenieria de software	se incluye conceptos como: herramientas y metodos de analisis economico para toma de desciones
Fundamentos Compuacionales, matematicos e ingenieriles	importancia del conocimiento de ciertas areas relacionadas con la ingenieria de software

Problemas típicos de la ingeniería software (Thayer, Oyster, & Wood (1981)

1. las especificaciones de los requisitos son frecuentemente incompletas
2. los proyectos a menudo se retrasan y exceden del presupuesto
3. no existen métodos para garantizar que el software entregado funcionara para el usuario
4. falta de detección sistemática de defectos de software
5. que no estén disponibles los procedimientos, métodos y técnicas para diseñar
6. relación poco predecible de la duración del proyecto
7. no disponibles mediciones o índices de “bondad”
8. hay cientos de miles de de métricas propuestas y no hay validez o aprobación

La globalización de los sistemas de extenso tamaño y contenido es una realidad actual.

Paradigmas emergentes	
Computación en la nube (Cloud Computing).	tipo de computación basada en Internet que proporciona recursos de procesamiento compartidos y datos, para computadoras y otros dispositivos, en base a demanda.
Computación social (Social Computing).	se refiere a los sistemas que permiten la obtención, representación, procesamiento, uso y difusión de la información que se distribuye a través de colectividades sociales tales como equipos, comunidades, organizaciones y mercados electrónicos
Datos masivos (Big Data).	conjuntos de datos que son muy grandes y complejos, tanto que las aplicaciones tradicionales de procesamiento son inadecuadas para manejarlos

Ingeniería de software:

Es básicamente la ingeniería que se encarga de la parte metódica de un proyecto… Un programador tiene que utilizar sus conocimientos para crear un programa o una aplicación que nos ayude o nos facilite un proceso o resolver un problema, pero el ingeniero en software tiene que analizar la situación y poder entenderla para aplicar métodos con los cuales poder crear un software que no solo resuelva el problema sino que sea además práctico, sin acciones innecesarias, y tener una idea de no solo crear en el proceso sino, poder crear con una base de lo que se quiere lograr.
La ingeniería de software, también, incorpora el análisis precedente de la situación, el bosquejo del proyecto, el desarrollo del software, el ensayo necesario para comprobar su funcionamiento correcto y poner en funcionamiento el sistema.

Objetivos de la ingeniería de software:
-Diseñar programas que se adecuen a las exigencias
-Liderar y acoplar el desarrollo de programaciones complicadas
-Actuar en todas las etapas de de vida del producto
-Computar los costos del proyecto y evaluar los tiempos de desarrollo
-Realizar el seguimiento de costos y plazos
-Liderar el equipo de de trabajo de desarrollo
-Documentar el funcionamiento correcto del proyecto/programa
-Incluir procesos de calidad
-Liderar y orientar a los programadores

Etapas de la ingeniería de software:

~ETAPA DE ANÁLISIS
En esta etapa se define con claridad el problema que habrá que resolver, para de esta forma saber qué es lo que se quiere lograr y cómo…

~ETAPA DE DISEÑO
Esta es una de las más importantes a mi gusto, pues aquí es donde uno deberá modelar o idear cómo será el producto al que se quiere llegar, y obviamente se debe tener en cuenta muchas cosas como el costo y el tiempo, establecer procesos, etc.

~ETAPA DE DESARROLLO
En esta etapa como su nombre lo dice, se desarrolla el modelo creado en la etapa anterior.

~ETAPA DE VERIFICACIÓN

Sirve para garantizar que efectivamente las características individuales del sistema cumplen con los requerimientos establecidos en la etapa de diseño.

~ETAPA DE IMPLEMENTACIÓN O ENTREGA
Es la etapa donde se entrega el producto final

~ETAPA DE MANTENIMIENTO
En caso de que se llegaran a encontrar errores o se pudieran adaptar mejores soluciones a ciertos problemas pues se haría una mejora al producto

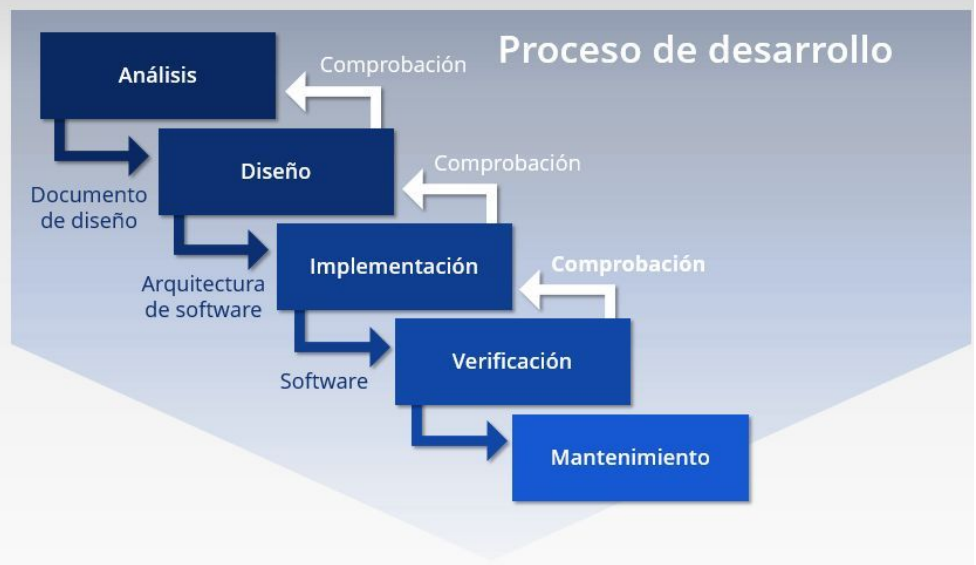
~ETAPA FINAL (EOL-END OF LIFE)
Consiste en ejecutar todas las labores que garanticen a clientes como empleados que el producto ya no estará en disposición.

Metodologías de Desarrollo de Software

Una metodología de software es un enfoque, una manera de interpretar la realidad o la disciplina en cuestión, que en este caso en particular correspondería a la ingeniería de software. De hecho, la metodología destinada al desarrollo de software se considera como una estructura utilizada para planificar y controlar el proceso de creación de un sistema de información especializada.
Entre los modelos existentes, algunos son:

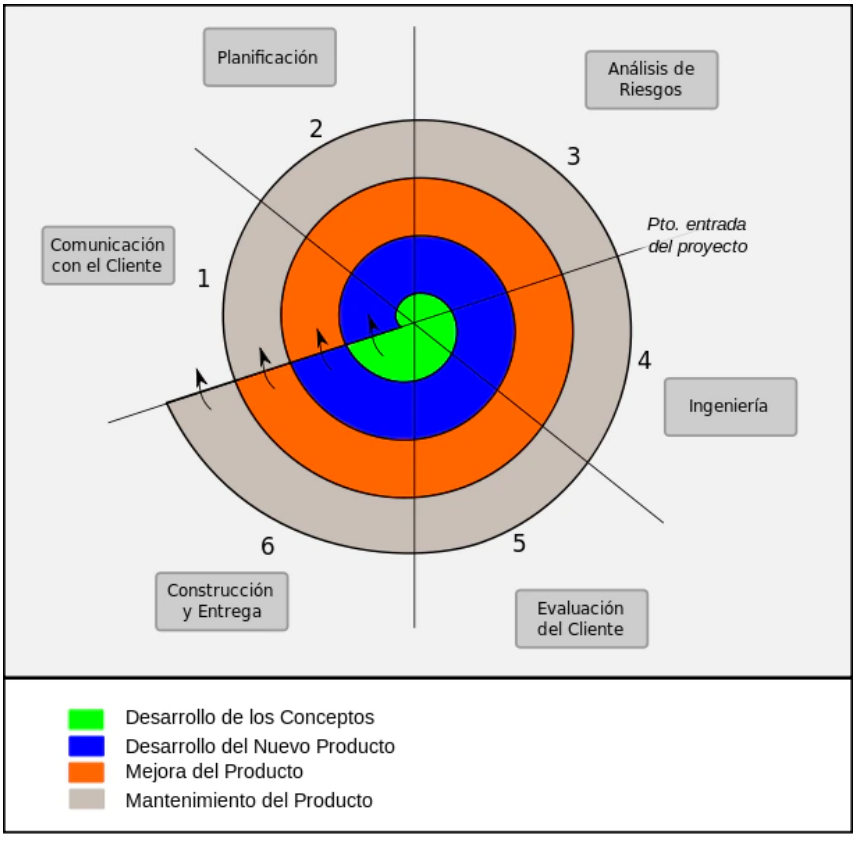
Modelo de cascada
En este modelo cada etapa representa una unidad de desarrollo con un pequeño descanso en medio. Por lo tanto, cada siguiente etapa inicia tan pronto como la anterior haya finalizado, y esos descansos son usados para confirmaciones de lado del cliente.
Podemos agregar que, este es considerado como el método tradicional de explicar el proceso de desarrollo de software en ingeniería de software, por lo que actualmente es visto como anticuado. Sin embargo, aún sigue siendo aplicado a proyectos con metas claras y requisitos que demandan hasta 100 hrs. de desarrollo, sobre todo considerando que este enfoque permite a los negocios deshacerse del papeleo innecesario, reuniones regulares que consumen mucho tiempo y retrasos en sus procesos de negocio.

Esta es una gran opción para pequeños proyectos donde todos los aspectos del proceso de desarrollo de software se conocen de antemano, pero una mala solución para proyectos complicados, ya que se trata de un modelo bastante inflexible.



Modelo de Espiral

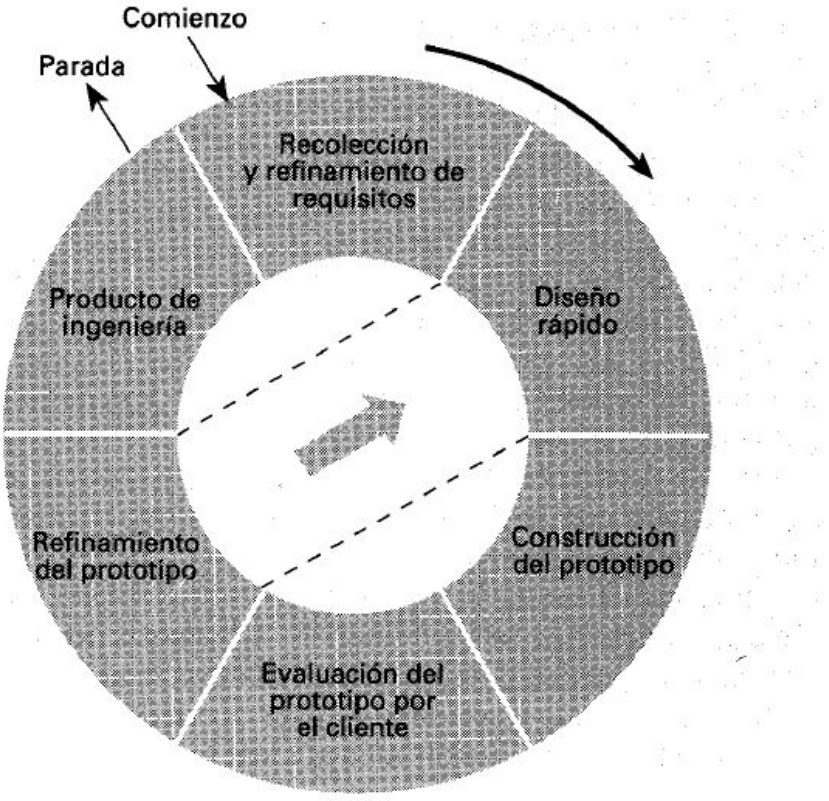
La metodología de espiral refleja la relación de tareas con prototipos rápidos, mayor paralelismo y concurrencia en las actividades de diseño y construcción. El método en espiral debe todavía ser planificado metódicamente, con las tareas y entregables identificados para cada paso en la espiral.



Metodología del prototipo.

Es un procedimiento de desarrollo especializado que permite a los desarrolladores la posibilidad de poder hacer la muestra de la resolución para poder validar su esencia funcional ante los clientes, y hacer los cambios que sean fundamentales antes de crear la solución final auténtica. De hecho, la mejor parte de esta metodología es que tiende a resolver un conjunto de problemas de diversificación que ocurren con el método de cascada.

Además de esto, la gran ventaja de optar por este enfoque es que da una idea clara sobre el proceso funcional del software, reduce el riesgo de falla en una funcionalidad de software y asiste bien en la recolección de requisitos y en el análisis general.

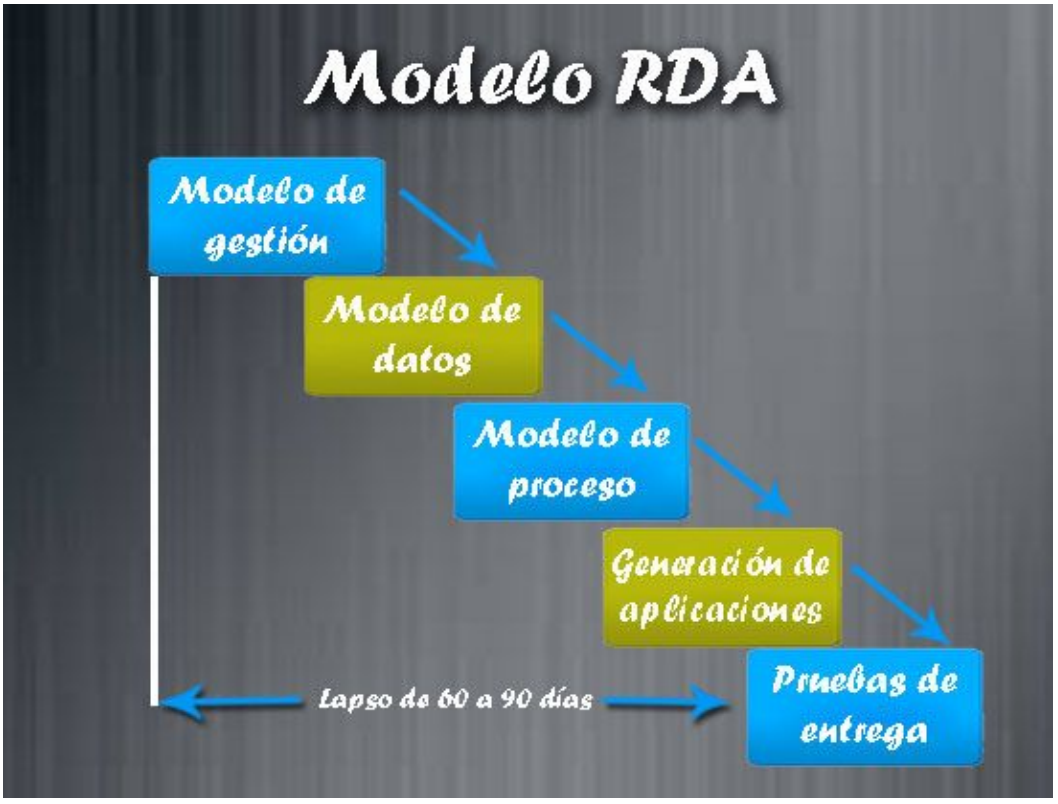


Desarrollo Rápido de Aplicaciones (RAD)

Con el objetivo de otorgar resultados rápidos, se trata de un enfoque que está destinado a proporcionar un excelente procesos de desarrollo con la ayuda de otros enfoques, pero además, está diseñado para aumentar la viabilidad de todo el procedimiento de desarrollo de software para resaltar la participación de un usuario activo.

Dicho esto, algunas de las ventajas a destacar de este tipo de desarrollo son las siguientes:

- Hace todo el proceso de desarrollo
- Asiste al cliente en la realización de revisiones rápidas.
- Alienta la retroalimentación de los clientes para su mejora.

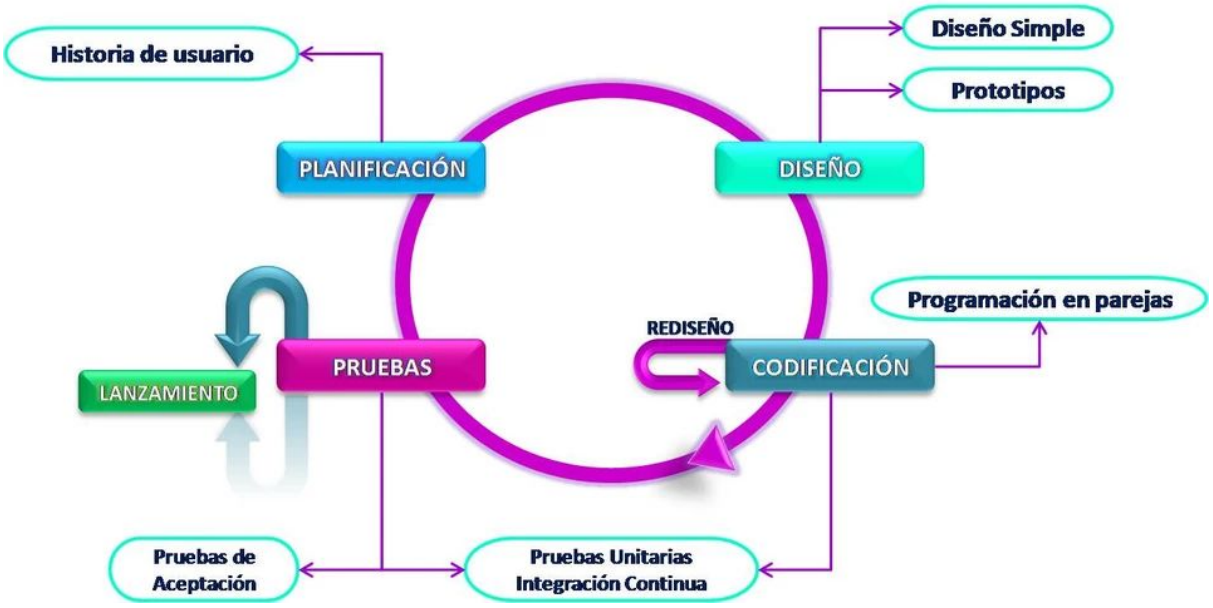


Metodología de Programación Extrema (XP)

Como metodología ágil de ingeniería de software, la metodología de programación extrema se conoce actualmente como metodología de XP (eXtreme Programming). Esta metodología, se utiliza principalmente para evitar el desarrollo de funciones que actualmente no se necesitan, pero sobre todo para para atender proyectos complicados. Sin embargo, sus métodos peculiares pueden tomar más tiempo, así como recursos humanos en comparación con otros enfoques.

TECNOLÓGICO NACIONAL DE MÉXICO®

PROGRAMACIÓN EXTREMA (XP)



Video 1

60 años de la computación en México

1. primera computadora 1958

En 1958 el Ing. Sergio Beltran trae la primera computadora al recién creado centro de Cálculo Electrónico de la Facultad de Ciencias, fue una IBM 650, con propósito general construida por IBM, pesaba 900 kg

Antecedentes del término ingeniería de software
Anthony Oettinger , presidente del ACM en 1966 habló de la profesión de la ingeniería en software.

Margaret Hamilton: 1965 dirige el desarrollo de software de navegación de apolo, uso el término de ingeniería de software para distinguir su trabajo de otras ingenierías.

Friedrich Ludwic Bauer: 1967 En el comité de ciencia de OTAN propone término ingeniería de software

Definicion de Ingenieria Software:

1. Aplicación sistemática de conocimientos científicos y tecnológicos, métodos y experiencia en el diseño, implementación, prueba y documentación de software.
2. Conjunto de programas de cómputo , procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computo

Conferencia de OTAN 1968
organizada: NATO SCIENCE COMMITTEE
localidad: Garmisch, Germany
Fecha: 7 al 11 de octubre 1968

Crisis de software:
se habló por primera vez del conjunto de dificultades o errores ocurridos en

1. proceso de desarrollo de software
2. calidad de software
3. costos
4. gestion
5. profesión

Problemas de proceso de desarrollo de software

1. falta de comprensión más completa del proceso de diseño del programa

2. construimos sistemas como los hermanos wright construyendo aviones

Video 2

Modelo de cascada
Herbert D. Bennington en 1956 define las bases de desarrollo como especializaciones
winston W. Royce en 1970 critica el modelo y propone mejoras y el prototipo
Bell & Thayer en 1976 introduce el término “waterfall”

Modelo de espiral
Barry Bohem en 1986 ciclos/iteraciones incrementales, con objeto, continuidad, basada en el análisis de riesgo.
Proceso unificado
Rational Software
Crearon Rational Unified Process RUP (tool) y Unified Modeling Language UML

Modelo agil
Scrum , Nonaka y Takeuchi 86(manufactura)

video1: manifiesto por el desarrollo ágil de software

se toma en cuenta	vs	no se toma en cuenta
individuos e interaciones		procesos y herramientas
software funcionando		documentacion excesiva
colaboracion con el cliente		negociacion contractual
respuesta ante el cambio		seguir plan

- Problemas de calidad

los problemas de lograr con fiabilidad suficiente en los sistemas de datos pueda cada vez más ser integrado en las actividades centrales de la sociedad moderna.
Alarmante son las fallas que a simple vista son inevitables de un software

- Logros de calidad de software

Modelo V, es un procedimiento uniforme para el desarrollo de producto de las TIC´s
Estándar utilizado para mejorar las actividades del proceso del software.

- Problemas de costos

Tj. Watson.
“os/360 costo a IBM gastó +50 millones de dólares al año”
Los costos de desarrollo del software son iguales a los costos de desarrollo de hardware
Se organizó una sesión especial sobre el tema de los precios de software en respuesta a la importancia de este tema en relación al futuro de la ingeniería en software.
mejoría: a favor de la fijación soportada de precios del software.

- Problemas de gestión.

dificultad de estimar/ampliar fechas y especificaciones en grandes proyectos software.
Identificar la naturaleza del progreso y la forma de medirlo.

Video2. continuación

- Problema de profesión

Acuerdo donde la ingeniería en software se encuentra en una etapa muy rudimentaria de desarrollo

CONFRONTACION	
ING. HARDWARE	INDUSTRIALES
ING. SOFTWARE	ARTESANOS

SIGLAS	DESCRIPCION	OFRECE
IaSS	infrastructure as service	maquinas virtuales y otros elementos de infraestructura
PaaS	platform as a service	previsiond e base de datos, servicios de web y otros entornos de ejecucion
SaaS	software as a service	prevision aplicaciones de software como servidores remotos

- Procesos de desarrollo

cambio profundo y radical métodos y teóricas
CONCEBIR, DISEÑAR,DESARROLLAR,PROBAR Y DESPLEGAR SOFTWARE.

- Calidad

la calidad y la seguridad del software se vuelven aún más importantes y críticas.

- Ambientes

Se determina el número de ambientes
se tiene como objetivo determinar el número de ambientes que un desarrollador debe considerar para la creación del software.

- Uso de energía

Debe diseñarse para minimizar el uso de recursos de hardware y comunicación, en consecuencia reducir el consumo de energía.
"*La calidad de nuestras vidas depende de la calidad del software, pero la calidad del software depende de la calidad de sus creadores y de las organizaciones que la respaldan"*

video3. SWEBOK.

- objetivos

promover una visión coherente de la ing. software
aclarar el lugar y establecer los límites de la ing software con otras disciplinas.

- Principios del proyecto

Transparencia: el proceso del desarrollo está en si mismo publicado y totalmente documentado.
creación de consenso, el proceso de desarrollo está diseñado para construir, a lo largo del tiempo, un consenso en la industria entre las sociedades profesionales y los organismos que establecen normas y en el ámbito académico.

Consenso sobre una lista de áreas de conocimiento
consenso sobre una lista de temas y materiales de referencia relevantes para cada conocimiento.

Consenso sobre una lista de disciplinas relacionadas.

Video 4.¿ Que tiene de apasionante la ingeniería software?

se tiene como objetivo que es realmente el ing software y la implementación de las mejoras del proceso de su desarrollo.
tecnología: conjunto de conocimientos acerca de técnicas que pueden abarcar el conocimiento.
existen dos variantes, tecnología incorporada y tecnología no incorporada.

- Software

Producto que diseñan y construyen

La ing. software es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción de software.

- Tipos de implementación

fórmulas preestablecidos
no todas las empresas son iguales
la mejor solución es aquella que mejor se adapte a la organización y salga de la agregación

- poca automatización

los procesos se caducan por lo que se necesitan darles mantenimiento
los procesos autorizados por alguna herramienta el mantenimiento se hace menos complicado.

- Pasión a la mejora.

Trabajar siempre en positivo
Trabajar buscando innovación
Trabajar en no arreglar errores, sino en optimizar las actividades
Trabajar en un ambiente agradable

VIDEO 1.

warren G.Bennis
nos ponemos ciertos pretextos para generar un proyecto.
la convivencia es fundamental para el desarrollo de software
*aplicaciones web
"aplicaciones" terceros
servidor web
app comerciales
sistemas operativos

tenemos sistemas cíclicos, no solo tema técnico
diferentes tipos de líderes y héroes
experimento científico o accidente
obtención de tecnología avanzada
traumas

RETOS DE LIDERAZGO
1- visión estratégica
¿cuando es el momento adecuado?
para empresa
proyecto
equipo
adaptación
2-¿cómo aprovechar lo que tengo?
equipo
conocimiento
infraestructura
tiempo
3-orientación a resultados
mas prevencion menos reacción
4-humanización del proyecto
conocer a cada miembro del equipo
motivar
unir
evitar la búsqueda de culpables
5- alimentar tu mente
adquirir nuevos conocimientos y habilidades
auto motivarse

ser parte activa de la empresa y de la industria
compartir su conocimiento y experiencia

El desarrollo de software es una gran oportunidad de desarrollarnos como personas somos valiosos cuando evitamos o mitigamos la deuda técnica

+documentación desactualizada, escasa, incompleta inservible o inexistente
errores no subsanados o desconocidos
control de versiones ineficiente o inexistente
desarrollo no escalable
problemas al incorporar nuevas funcionalidades
dificultades a la hora de actualizar la tecnología o migrar a una nueva plataforma.

INNOVACIÓN
innovamos cuando renovamos y ampliamos la gama de productos y servicios
renovación o creación de procesos productivos
guiando y formando líderes e innovadores

VIDEO2.dispara el motor de la visión en tus proyectos con innovación
“llegar al mismo objetivo pero con beneficio a la organización”
innovación empresarial se ha convertido en tema común ya que profesionistas y empresarios están buscando mejorar sus modelos de negocio y esto conlleva realizar grandes cambios organizacionales y tecnológicos para poder adaptarse fácilmente a los cambios del mercado, desarrollar ágilmente productos y servicios, tener visión a largo plazo y crear ventajas competitivas sumamente fuertes.

Para innovar, los profesionistas emplean mejores prácticas en compañías de todos tamaños. La innovación no es opcional, es obligatoria para quienes buscan mejorar desempeños actuales en ambientes complejos.

tomar la información y poderla aplicar.

clasificar la información
lluvia de ideas

post it
clasificar ideas
obtener complementos importantes
llamado método de brainstorming
para iniciar la innovación es importante realizar las siguientes preguntas
¿Qué pregunta asesina te gustaría realizar? no te limites
¿cual es la regla mas estupida que tienes en tu organización?, cambiala
¿cual es la mejor manera para facilitar el cambio usando diversidad colaborativa? involucra mayor cantidad de personas

VIDEO 3.Principios de Negocio para personal de IT

ofrecer un prodcuto y este sea pagado por ese producto y mi empresa se beneficie
qué hacer con la idea
cómo materializar esa idea, que soporte nóminas, costos, beneficios, tenga permanencia, y se mantenga en el mercado.
la innovación no es un lujo intelectual

solo el 11% de las campañas fortune 500 de 1955 existen en la actualidad, mientras que el tiempo promedio que las empresas se quedan en el top 500 se ha reducido de 75 años a 5.Los países con gobiernos sin horizonte pueden correr la misma suerte que las empresas obsoletas. La elección es simple y conocida: innovar, o ser irrelevante.

ESTRATEGIA DEL OCEANO AZUL

“un desafio para que las compañias abandonen el sangriento oceano de la competencia y creen espacios seguros en el mercado, en los cuaes la competencia no tenga importancia”

W.Chan Klim y Rene Mauborgne

OCÉANOS ROJOS

El mundo de los negocios se ve representado en su mayoría por el océano **rojo**, regido por la competencia entre las empresas. Por el contrario, la estrategia del océano azul busca ampliar el mercado a través de la innovación. ... En primer lugar, el océano **rojo** se refiere a la alta competencia

OCÉANOS AZULES

Se define como aquel espacio perteneciente al mercado y que aún no ha sido utilizado o explotado, y que por consiguiente generará una oportunidad para el crecimiento rentable, que tiene muchas más ventajas.

METODOS PREDICTIVOS

Resultados de metodos predictivos
solo el 32 % de los proyectos de IT se entrega razonablemente a tiempo y bajo presupuesto
44% de los proyectos son considerados “retados”(tarde, sobrepresupuesto, o con funcionalidad equivocada)
Aproximadamente 1 en 4 (24%) de los proyectos se consideran fallidos
En promedio, los proyectos por fuera de presupuesto se desvian un 189% del costo estimado
Las estimaciones de Software hechas sin detalles completos presentan errores que varian en promedio entre 85 y el 772%

MÉTODOS ADAPTATIVOS

AGILE ENTERPRISE

Empresa rápida, flexible y robusta capaz de responder rápidamente a los retos, eventos y oportunidades inesperados.

Framework de Innovación

Agile Enterprise

ACTIVIDADES semana 7 (Nov 2-6, 2020)

VIDEO1. Lunch & Learn Webinar: Análisis de Negocio Ágil, ¿es esto viable?

Agilidad(Agile) es un término utilizado para describir una serie de metodología para el desarrollo iterativo de software, que se han desarrollado a lo largo del tiempo.
Algunos rasgos comunes entre metodología agile son: liberación frecuente de productos; altos niveles de colaboración del equipo en tiempo rea; documentación reducida y evaluación frecuente de riesgos y valor de negocio.

Manifiesto Agil
Individuos e interacciones sobre procesos y herramientas
Software funcionando sobre documentación extensiva
Colaboración en el cliente sobre negociación conceptual
respuesta ante el cambio sobre seguir un plan

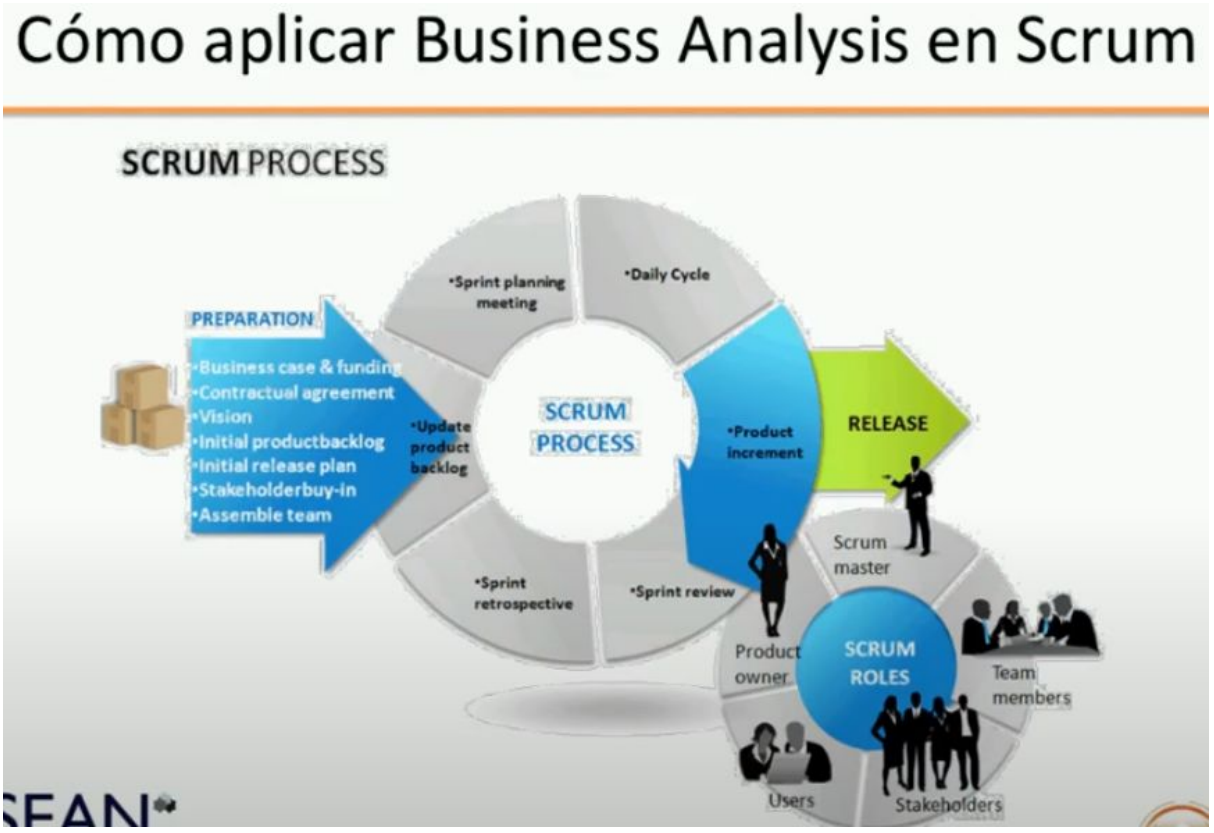
Casi todas las metodologías de administración de proyectos se ubican en algún lugar a lo largo del espectro entre los enfoques de la 'administración basada en planes' y de la 'administración basada en cambios'

Administración basada en planes

Objetivo	asegurar que la solucion esta completamente definida antes de su implementacion
Analisis	Ocurre al inicio del proyecto o durante una fase especifica del mismo

los cambios solo ocurren cuando son genuinos y están claramente justificados.
un objetivo: rapida entrega de valor de negocio en iteraciones cortas
el analisis: lista inicial de requerimientos de alto nivel
Gestión de cambios
Generalmente no existe un proceso de gestión de cambios distinto de la selección de requerimientos de una interacción dada.

Comunicación: Se enfoca mas en una comunicación frecuente que en una comunicación formal



Tenicas Scrum

Tecnicas Scrum mas importantes	
Blocklog Management	Este es el metodo principal para gestionar tanto la piorizacion de los requerimentos como el control de cambios en la mayoria de los metodos agiles
restrospectives	Esta es una practica comun utilizada por los equipos agiles para mejorar su metodo de trabajo.Los analistas de negocio deben de buscar retroalimentacion de los requerimentos que proporcionan al equipo y como y cuando esos requerimientos son suministrados , con el fin de encontrar formas de mejorar sus procesos

VIDEO 2. Inteligencia de negocios, liderazgo y toma de decisiones: la tercia perfecta para la competitividad
Lograr que las empresas sean de alto rendimiento
El mundo actual ronda entorno a los negocios, por lo tanto lo que tenemos en el mundo tenemos negocios
primero debemos entender la evolución

- como las especies luchan unas con otras, para adaptarse, a esta competencia se llama selección natural, lo agregamos en el contexto empresarial, es la formas en cómo las empresas se adaptan y son competitivas en su rama
- empresas cambian de logo, slogan, línea de servicio y tipo de servicio

Hoy en dia hablamos de empresas 2.0 debido a la web 2.0
utilizan información para poder crecer.
NEGOCIO:
actividad sistema, Método o forma de obtener un beneficio (generalmente monetario) a cambio de ofrecer bienes o servicios a otras personas

Es la negación del ocio (Neg-ocio).

Lo necesario para emprender un negocio:
CAPITAL
EXPERIENCIA

capital pero no experiencia: inversionista
no capital y experiencia: emprendedor
capital y experiencia: empresario
Etapas elementales de los negocios:

- ¿Que? viene predeterminado o bien se determina en el ¿Donde?
- ¿Dónde? análisis y estrategias de mercadeo
- ¿Como? modelo de negocios, inteligencia analitica
- ¿Cuando?
- Análisis factibilidad, prospección temporal.

la inteligencia de negocios : uso de conocimiento e información por medio de tecnologías de información IT, para emprender o mejorar un negocio
y tomar las mejores decisiones

Inteligencia analítica

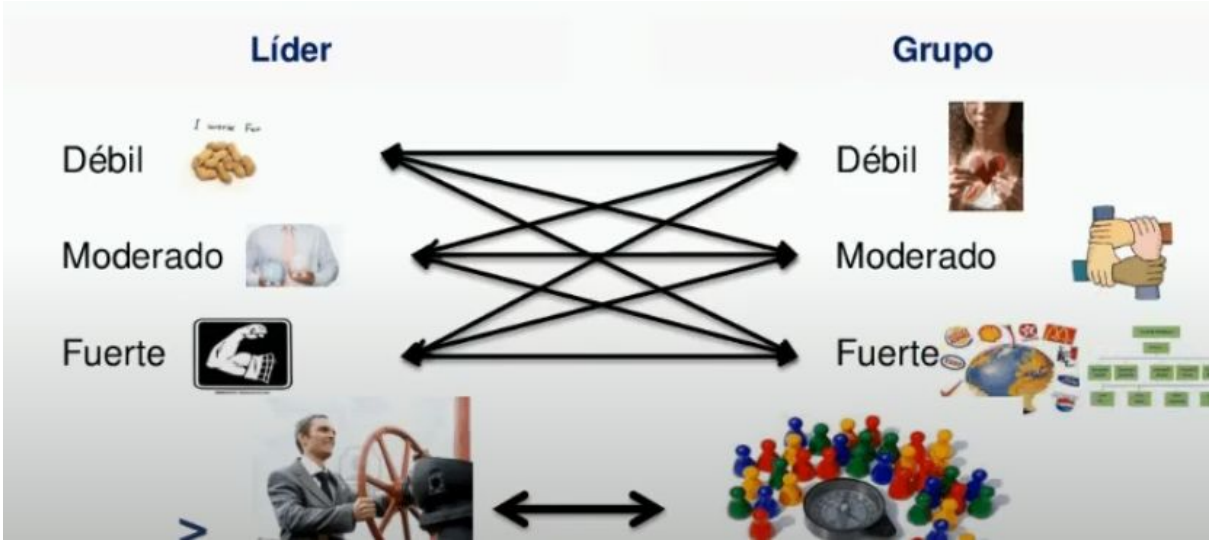
1. Estadísticas de Datos
2. Análisis de tendencias
3. pronósticos
4. modelos explicativos
5. optimización de recursos

Tipos de liderazgo

TECNOLÓGICO NACIONAL DE MÉXICO®

INSTITUTO TECNOLÓGICO DE IZTAPALAPA®

TEORIA DE LÍDERES



Leyes de las decisiones

1.Toda decisión tomada es irreversible

la decisión tomada fue la óptima de acuerdo con algún o algunos criterios los criterios cambian con el tiempo , por lo que las decisiones siempre estarán mal de acuerdo a algunos criterios.

TIPO	DESCRIPCION
ANALITICA	ANALIZA TODAS LAS OPCIONES POSIBLES, LOS PROS Y LOS CONTRAS EN DIVERSOS SENTIDOS(ECONOMICOS PRINCIPALMENTE), PERO SIEMPRE DE FORMA CUANTITATIVA(MEDIBLE), POR LO QUE NO SE DEJA LLEVAR POR CUESTIONES CUALITATIVAS QUE INVOLUREN SENTIMIENTOS, NO CAMBIA DE DECISION TOMADA A MENOS QUE SE RECURRA A UN CRITERIO DIFERENTE
IMPULSIVA	TIENDE A DEJARSE LLEVAR POR LA SITUACION Y TOMAR DECISIONES EN BASE A ELEMENTOS SENTIMENTALKES MUY CUALITATIVOS, NO CAMBIA LA DECISION TOMADA, PUESTO QUE ESO ROMPE CON SU MANERA DE REITERAR SU FORMA DE SER IMPULSIVA
EXPRESIVA	TIENDE A DEJARSE A LLEVAR POR LA SITUACION Y TOMAR DECISIONES INFLUIDO POR OTROS , SIN ANALIZAR LOS PROS Y CONTRAS DE CADA POSIBLE RESULTADO, DE FORMA CUALITATIVA PRINCIPALMENTE, CAMBIA MUCHO DE DECISION O BIEN NO LA TOMA CON EL TIEMPO ADECUADO, PUESTO QUE ESTA PERSONALIDAD TIENE COMO CARACTERISTICA LA DESIDIA
PASIVA	ANALISA LAS OPCIONES POSIBLES USANDO CRITERIOS CUANTITATIVOS Y CUALITATIVOS, DANDO MAS IMPORTANCIA A LAS CUESTIONES MEDIBLES COMO ECONOMICAS, PRODUCTIVAS, ETC. TIENDE A NO DECIDIRSE O BIEN A CAMBIAR DE DECISION POR PENSAR NO HABER TOMADO LA OPCION MAS ADECUADA

Las decisiones analiticas

1. definir el problema
2. Analizar el problema
3. Evaluar las alternativas
4. Elegir las alternativas
5. Aplicar la decisión

“Un buen líder con un buen liderazgo y buen equipo de trabajo toman las mejores decisiones y llevan al máximo un buen negocio”

“Los líderes piensan y hablan sobre soluciones.Los seguidores piensan y hablan sobre problemas”

Video 3. Modelos de negocios relacionados a aplicaciones móviles
modelo de negocio: como hacer dinero
factores que ayuden a generar dinero, si el modelo no da resultado de dinero, no es un buen modelo de negocio.
Modelo de negocio

- La forma en que creará valor para el cliente intercambiando ese valor por un beneficio para mi empresa

1. definición y perfilamiento de mercado
2. Definicion de cliente
3. Propuesta de valor
4. Estrategia llegada al cliente
5. Esquemas de distribución del producto
6. Diferenciadores de mi propuesta de valor
7. “Uniqueness” de mi propuesta de valor
8. Integración de mi cadena de valor
9. ESQUEMAS DE RELACIONAMIENTO CON EL CLIENTE

- Esquemas elementales de relación con el cliente

1. Suscripción
2. Renta
3. Licenciamiento
4. Servicios

no solo depende de la aplicación en el punto virtual que desarolle
mucho seguirá pasando por lo servicio que pueda entregar en el “mundo real”.
INDUSTRIAS MADURAS, ACTIVIDADES EMERGENTES

- Escenarios en industrias de manufactura

1. Acciones y seguimiento en la línea de producción
2. Flujo y aprobación de informacion
3. Inspección y aseguramiento de calidad
4. Logistica y distribucion
5. Entrega y distribución al cliente
6. Conocimiento del comportamiento del producto en el mercado.

- Retail

1. Conocimiento del comportamiento del cliente en el piso de ventas.
2. Reconocimiento y preferencias del cliente para la creación de nuevas necesidades

3. Monitoreo y seguridad de mercancía en el piso
4. Control de inventarios y ciclo de venta de productos
5. Mejorar las experiencias de compra del cliente
6. Esquemas de fidelización

- Servicios

1. Mejorar la interactividad entre el establecimiento o prestador de servicio y el cliente
2. Automatizar el proceso de captura de necesidades del cliente
3. Esquemas de fidelización de clientes
4. Proporcionar información y conectarla con la decisión propia.

- Sistemas alimentarios

1. Sistemas de cultivo y monitoreo de crecimiento
2. Proceso de trazabilidad
3. Inocuidad alimentaria

- Salud pública

1. ECE
2. Inventarios y disponibilidad de servicios y productos
3. Monitoreo y seguimiento de pacientes
4. “Entretenimiento”

- Transporte público

1. Ubicación
2. Identificación
3. Monitoreo de traslado
4. Estimaciones de tiempo y ruta

- Sociedad en su contexto

1. Interacción
2. Relación
3. Esparcimiento
4. Alerta temprana
5. Contingencia ambiental
6. Desde mi sillón en el contexto de lo que escucho y veo

- Teléfonos inteligentes, autos inteligentes

1. Servicios de asistencia remota
2. Monitoreo remoto de ubicación y ruta pero también de condiciones generales
3. Interacción en red con otros vehiculos y mobiliario urbano
4. Proveer información en ruta cerca del contexto de servicios
5. Entretenimiento a bordo del vehículo

(ACT. SEM 8 NOV9-NOV13, 2020)

VIDEO 1. Aplicando los 12 principios del manifiesto ágil a la gestión de tus proyectos

Agilidad: capacidad para adaptar el curso del desarrollo a la evolución de los requisitos y las circunstancias del entorno de los proyectos.

METODOLOGIAS AGILES
EXISTEN REQUISITOS DESCONOCIDOS O VARIABLES
SE TIENE NECESIDAD DE UNA RAPIDA IMPLEMENTACION
CUANDO SE REQUIEREN ENTREGAS DE VERSIONES PREVIA A LA ENTREGA FINAL
EXISTE ALTA PROBABILIDAD DE REALIZAR CAMBIOS
EL EQUIPO DE DESAROOLO ES DE 3 A 8 PERSONAS
LA EMPRESA DONDE LABORAS NO TIENE JERARQUIAS ESTRICTAS Y ALTO GRADO DE CULTURA CEREMONIOSA

1. SATISFACCION DEL CLIENTE

nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor

2. Cambio

Aceptamos que los requisitos cambien incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

3. Software funcional

Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
la naturaleza de los métodos ágiles nos permite el desarrollo iterativo de versiones, estableciendo el beneficio de la entrega de aquella funcionalidad que es crítica para el cliente
la estrategia debe estar alineada a los objetivos de negocio del cliente y de preferencia ser identificados desde el anteproyecto.

4. Trabajo en equipo

Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
Los miembros de un equipo ágil establecen un objetivo común cuando entre todos deciden cuantos requisitos/objetivos son capaces de completar en una sola iteración
Debe existir compromiso conjunto al elaborar la estrategia

5. Motivación

Los proyectos se desarrollan entorno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiables en la ejecución del trabajo.

las personas están más motivadas cuando pueden usar su creatividad para resolver problemas y cuando pueden decidir organizar su trabajo.
Las personas se sienten más satisfechas cuando pueden mostrar los logros que consiguen.
Crean vínculos

6. Comunicación efectiva

El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
es importante generar tantas oportunidades de comunicación bidireccional como sea posible.

7. Medir el progreso

El software es la medida principal del progreso
Revisar el trabajo realizado el dia anterior y el previsto para el dia siguiente
Identificar las tareas donde el equipo está teniendo problemas y no avanza para poder tomar decisiones al respecto.
Cada miembro del equipo debe medir su tiempo y tomar decisiones objetivas sobre el uso

8. Ritmo del equipo

Los procesos ágiles promueven el desarrollo sostenible. Los promotores desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida
Cada equipo tiene su ritmo y es responsabilidad de todos medirse y ajustarse

9. Atención y diseño

La atención y el diseño continua a la experiencia técnica y al buen diseño mejora la agilidad
Una vez que tenemos el diseño, podemos planear la construcción
El software el énfasis que más valor aporta es el de análisis y diseño de modo que requiere de personas creativas y talentosas.

10. Simplicidad

La simplicidad o el arte de maximizar la cantidad de trabajo no realizado es esencial.

Evita desperdicios:

- ❏ Código o funcionalidad innecesarias
- ❏ Empezar más de lo que puede terminarse
- ❏ Requerimientos poco claros o con cambios constantes
- ❏ burocracia
- ❏ comunicación lenta o inefectiva
- ❏ trabajo parcialmente terminado
- ❏ Quita tiempos personales

11. Auto-organización

Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados

Margen de decisión suficiente para tomar las decisiones que consideren oportunas.

Los líderes deben valorar las ideas y opiniones de su equipo y demostrarlo mediante el uso de su posición para ayudar al consenso de la forma de proceder, en lugar de decirles simplemente que hacer.

12. Reflexión

A intervalos regulares el equipo reflexiona sobre como ser mas efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia

No tener miedo a tomar decisiones

No tener miedo a cambiar una decisión temprana: cuando parezca nueva información, no debemos tener miedo a cambiar la decisión

Analizar en conjunto si nuestros métodos pueden mejorar.

Para hacer la diferencia debemos de cambiar nuestro énfasis ¿Dónde pones tu énfasis actualmente?

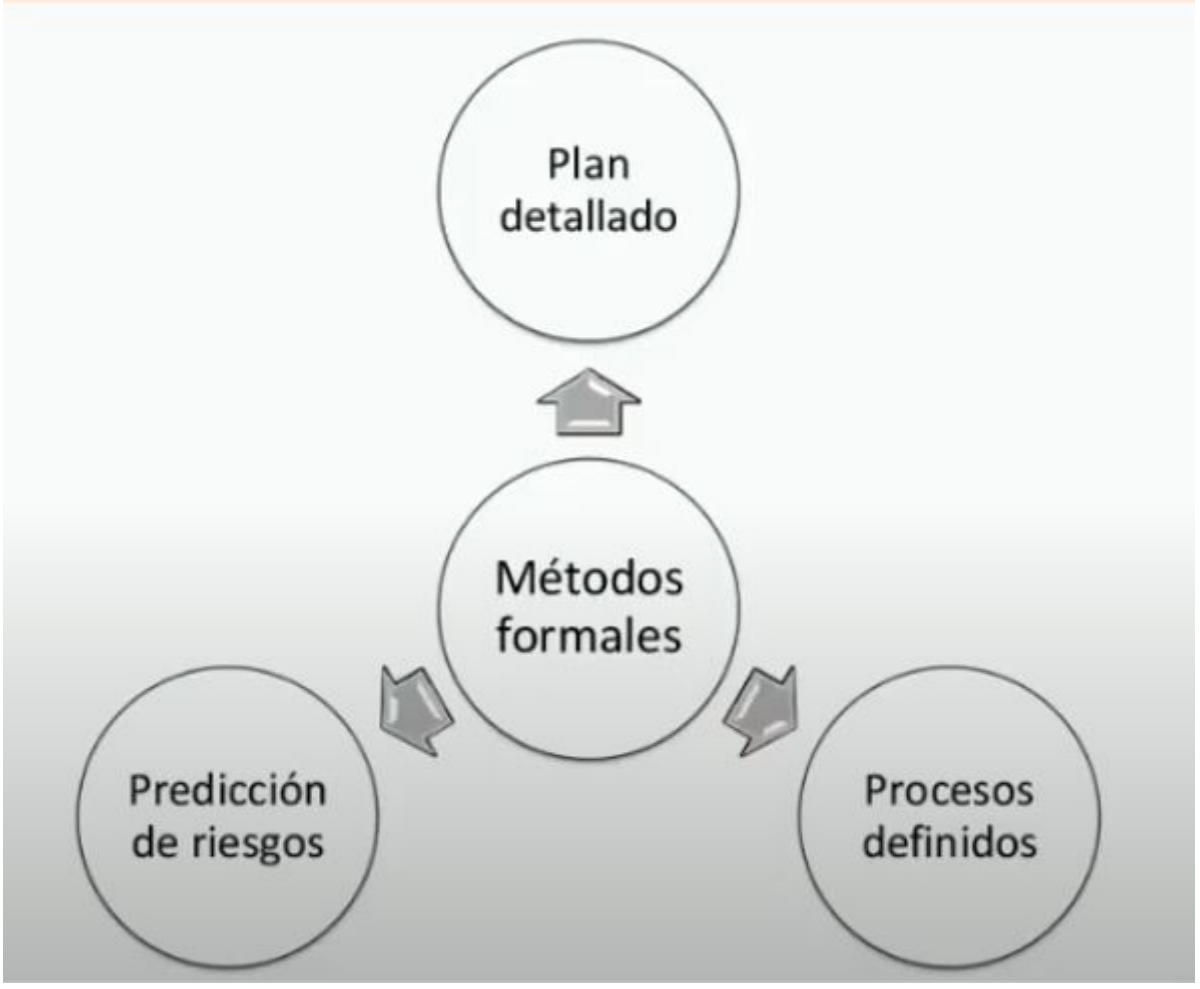
La metodología no es lo único : El liderazgo es vital.

Decidir que las personas son primero es una decisión grande, que requiere mucha determinación

El desarrollo ágil de software pone el énfasis en el factor más importante: el factor humano.

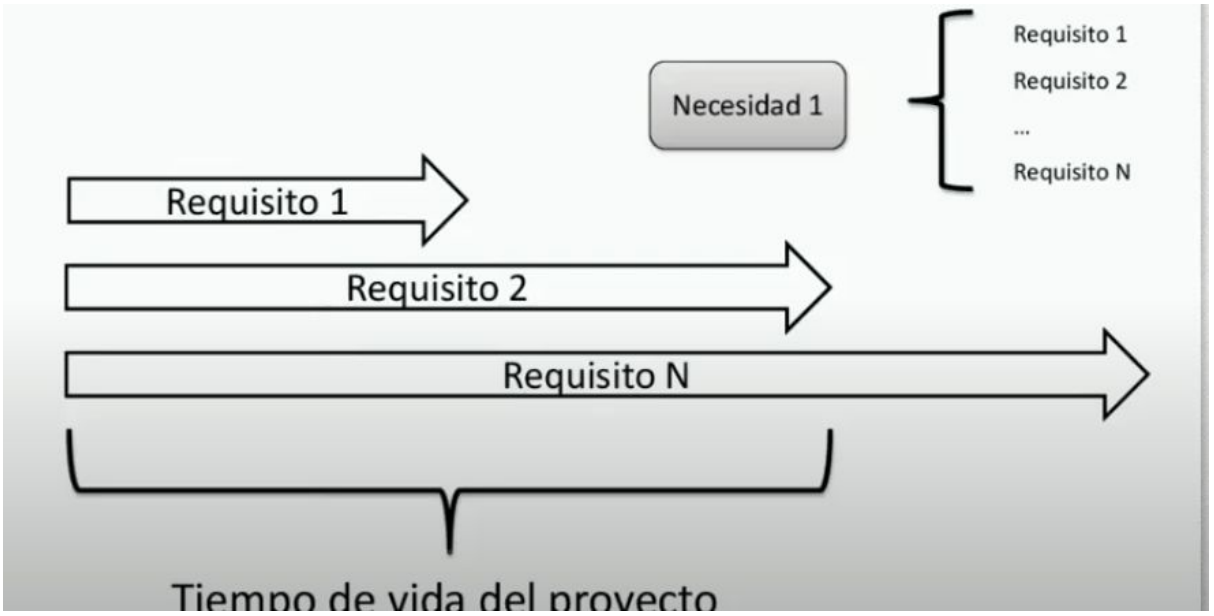
VIDEO 2.EL FUTURO DE LA AGILIDAD EN SOFTWARE

Características de los métodos formales



Carencias de los modelos formales

- Incapacidad para gestionar los cambios de requerimientos
- No consiguen involucrar a los usuarios finales
- Un proyecto no es seguir una serie de pasos, es construir un entregable
- Primar la cultura del cumplimiento }, confunde “hacer el plan” con “hacer el producto”
- No sabe cómo administrar el talento , la creatividad, lo “mejor” frente a lo “correcto”
- Fallan en tratar de predecir los riesgos
- Es una actividad caótica: “Codifica y corrige”



CALIDAD EN EL DESARROLLO DE SOFTWARE

- En la actualidad no existe “Calidad” en el mundo del desarrollo de software
- Los métodos de calidad de software se basan en “cómo” hacemos, no en “que” hacemos.
- La calidad del producto no depende solo de la calidad del proceso
- No hay opinión clara sobre cómo medir la satisfacción del cliente

A los individuos y su interacción, por encima de los procesos y las herramientas.

- Los procesos sirven para fijar “a posteriori” no “a priori”
- El proceso mata la creatividad y la innovación
- Fomentan una cultura del cumplimiento, el “como” antes que el “que”.

El software que funciona por encima de la documentación exhaustiva

- Un cliente necesita software, no documentos
- La documentación se usa como barrera de responsabilidad
- Fomenta construir lo correcto, en lugar de construirlo correctamente.

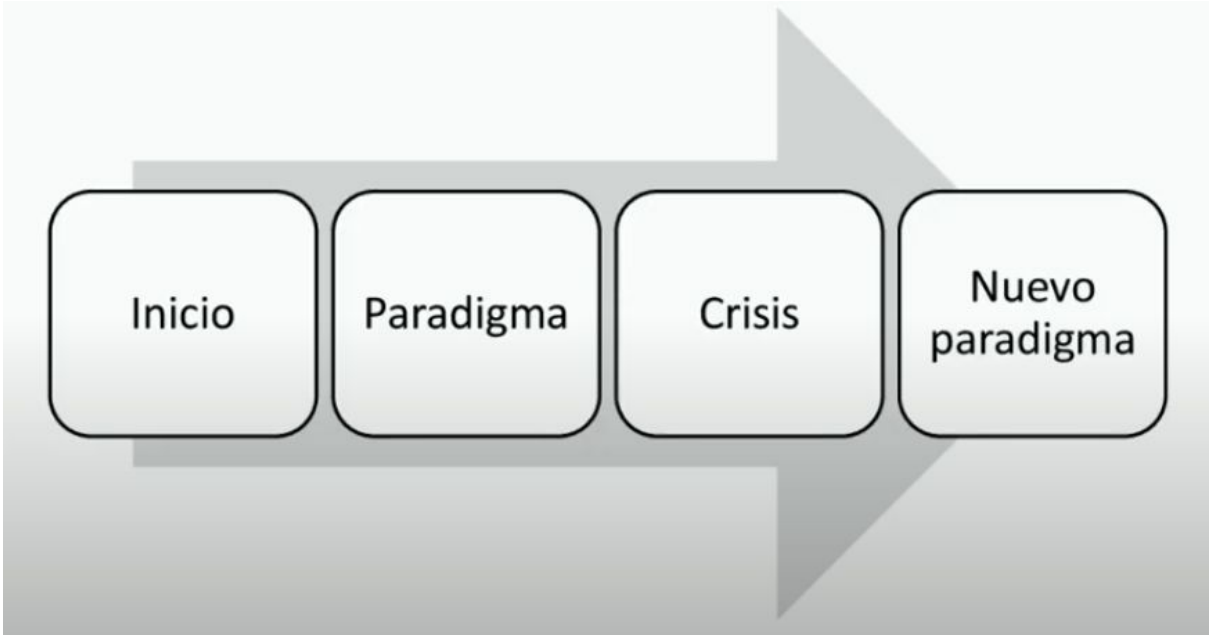
La respuesta al cambio, por encima del seguimiento de un plan

- En entornos altamente inestables, la predicción del futuro es un ejercicio fútil
- Los planes altamente estructurados y definidos no garantizan un seguimiento adecuado .
- Gestión de avance y riesgo mediante la adaptación (control empírico)

Lo que no es agilidad:

- No tiene actividades específicas de diseño de arquitectura
- Los proyectos los hacen equipos, los equipos los componen personas y las personas no siempre hacen lo que deben
- Ignora la importancia de la relación contractual
- Difícil de aplicar en grupos grandes y estructurados
- Gestión de los alcances y necesidades cuando el proyecto es grande (¿Cuánto cuesta?)

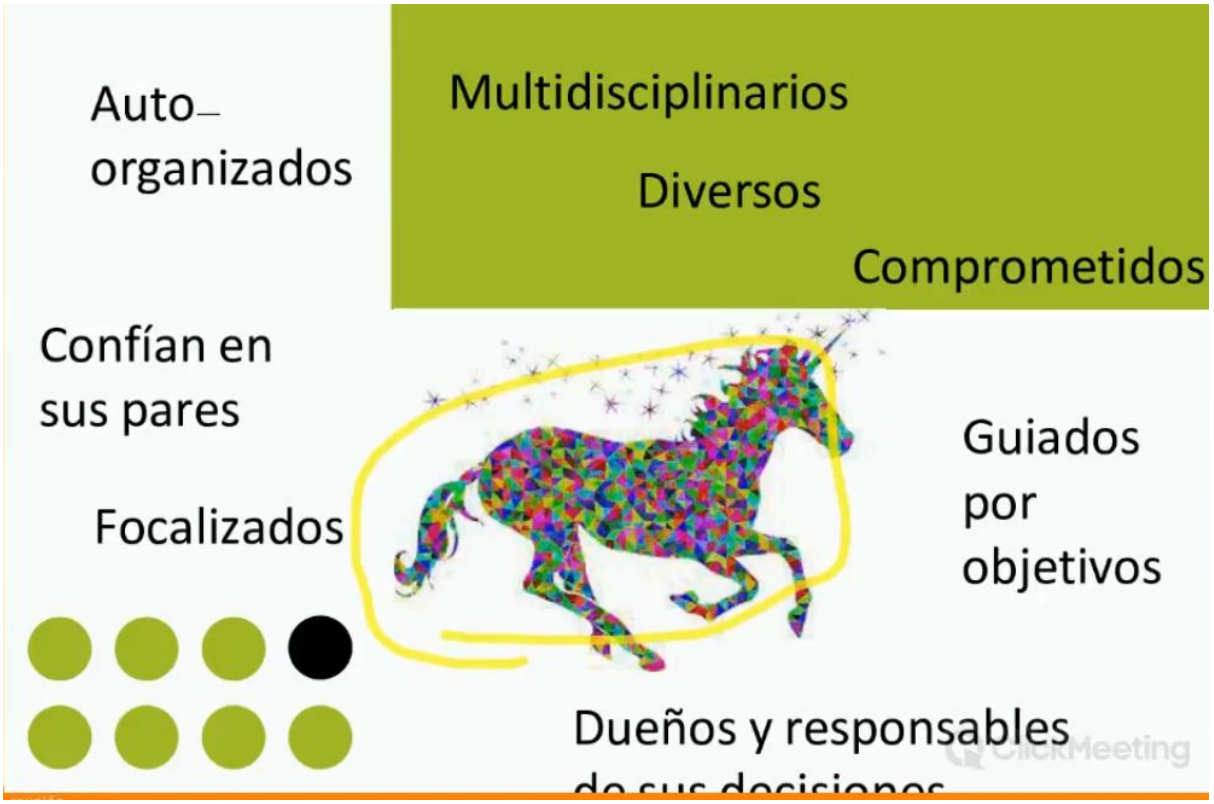
ETAPAS KUHN



CUESTIONES A DESARROLLAR

VIDEO 3. EL AGILE MINDSET: MÁS ALLÁ DE UNA METODOLOGÍA
Las aplicaciones cambian y se requiere una adaptación ligada a la innovación
¿Cómo hacemos las cosas?
Tener un impacto al mundo
regidos por la innovación buscamos ganar y tener un impacto en la sociedad, se moldea nueva sociedades

Los manager deben nutrir a la creatividad
Cada integrante de un equipo debe tener sus propias habilidades las cuales deben de ofrecer en el quipo



Realizar equipo auto organizado, con diversidad, sean focalizados, guiados por objetivos.
Dueños y responsables de sus objetivos.

Transferencia: impulsa liderazgo, colaboración, reflexión,comunicación efectiva ser un agente de cambio .

ACTIVIDADES SEMANA 10(NOV 23-27, 2020)

VIDEO 1.COMBATIENDO BARRERAS DE PRODUCTIVIDAD A TRAVÉS DE PLANEACION PRÁCTICA PARA DESARROLLOS DE SOFTWARE

Con demasiada frecuencia el desarrollo de software se enfrenta a la batalla de navegar en contracorriente con respecto al tiempo
usualmente malas prácticas de estimación, comunicación y administración del tiempo.
Se debe basar un proyecto en los tres principales pilares

- Estimación
- Administración de tiempo
- Comunicación

DEBILIDADES EN ESTIMACION QUE REDUCEN TIEMPO	
ESTIMACION POR TRAMITE CON ENFOQUE	"haremos todo lo posible por salir en esas fechas"
NO TENER METODOS PARA ESTIMAR	Usar el juicio experto es un metodo, no una adivinanza
NO TENER UNA ADECUADA PRIORIZACION	Todo siempre es urgente"

Debilidades en comunicación que reducen tiempo:

- Iniciar sin haber identificado todos los requerimientos críticos

Una vez identificados es necesario evaluar si el equipo está listo para afrontarlos y determinar la manera en la que se va afrontar .

- Delegar sin dar tiempo a explicaciones claras sobre el alcance

Fomentar la colaboracion y comunicacion abierta nos brinda agilidad

- No saber decir “no”. Decir “no” puede ser una de las herramientas de administración de tiempo más poderosas que puedes llegar a dominar.

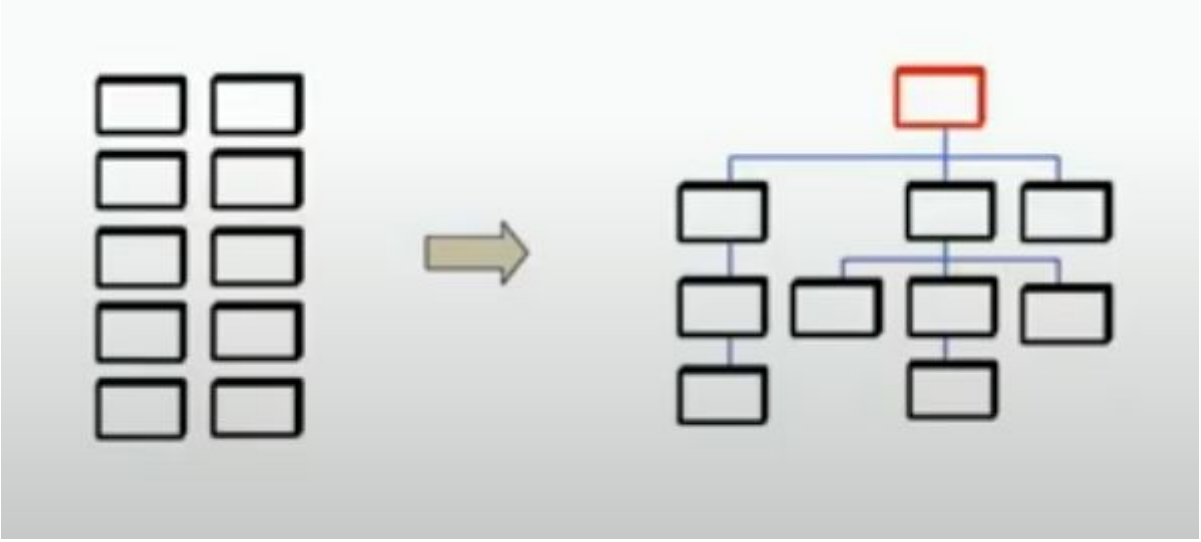
Ser asertivo. Piensa que no tienes obligación de cumplir deseos de los demás, confía en que puedes hablar con claridad con las personas y ellos entenderán tus razones o podrás negociar tiempo.

VIDEO 2. PEQUEÑO TALLER DE ADMINISTRACIÓN DE PROYECTOS
un proyecto podría definirse como un conjunto de actividades que desarrolla una persona o una entidad para alcanzar un determinado objetivo.

Un elemento importante en un proyecto es el líder de proyecto, quien es el responsable de detectar las necesidades de los usuarios y gestionar los resultados económicos, materiales y humanos para obtener los resultados esperados en los plazos previstos y con la calidad necesaria

Para realizar un proyecto se requiere de un equipo de trabajo y una estrategia Una estrategia son el conjunto de acciones que alinean las metas y objetivos de un proyecto, son acciones muy meditadas , encaminadas hacia un fin determinado.

En la descomposición jerárquica del trabajo a ejecutar por el equipo (no por el líder) orientado a productos entregables para alcanzar los objetivos de un proyecto



Balanceo dinámico de cargas

- Planeación por paquetes.
- Tareas no mayores a 40 horas.
- Tareas con duración de una semana pero el esfuerzo en horas, las horas que se le van a aplicar en la semana.
- las dependencias entre paquetes no sobre tareas.

Matriz de Rastreabilidad

La matriz de rastreabilidad, es una tabla que vincula los requisitos con su origen y los monitorea durante la evolución del proyecto.

Cambios de alcance

- Implicaciones técnicas y tecnológicas
- Insumos
- Tiempo en calendario
- Esfuerzo (horas)
- Cual es el mejor momento para desarrollar el cambio

SEMANA 11(30 NOV-4 DIC)

VIDEO 1.

muestra cómo usar las herramientas de creación de prototipos para vencer las probabilidades y asegurarse de que sus ideas tengan éxito en el mercado. Con docenas de ejemplos y estudios de casos, conocimientos de su tiempo en Google y Stanford, así como su experiencia como emprendedor e innovador, The Right It está lleno de herramientas, técnicas y tácticas poderosas, efectivas y de acción inmediata.

Prototipado en Stanford

Vea cómo Alberto Savoia explica The Right It, Pretotyping y Seven Strategies para combatir las fallas del mercado en el Stanford Entrepreneurial Thought Leaders Seminar.

Las siete estrategias:

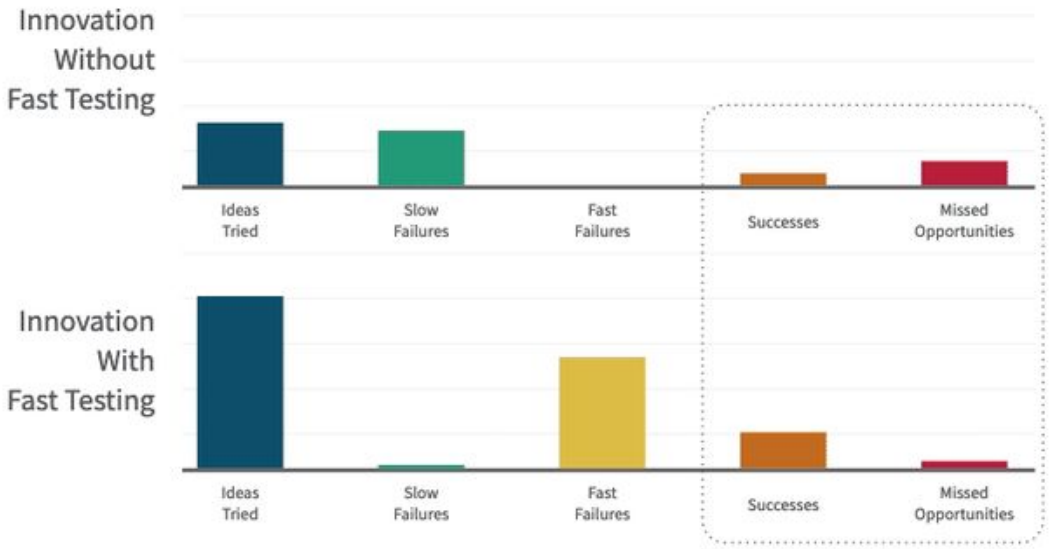
1. Obedezca la ley de las fallas del mercado.
2. Asegúrese de estar construyendo The Right It.
3. No se pierda en Thoughtland.
4. Confíe solo en sus propios datos (YODA).
5. Pretotipar.
6. Dígalo con números.
7. Piense globalmente, pruebe localmente.

La mayoría de las nuevas ideas fracasan en el mercado, incluso si se ejecutan de manera competente.
A esto le llamamos la Ley de fallos del mercado.

Solo hay una forma de luchar contra la ley de las fallas del mercado: pruebe el mercado en busca de sus ideas de manera objetiva, rigurosa y rápida antes de invertir para desarrollarlas.

Pretotyping le proporciona las herramientas y técnicas que necesita para validar su idea con recursos mínimos y en muy poco tiempo (tan solo unas pocas horas).

Con la creación de prototipos, puede permitirse probar muchas ideas nuevas e innovadoras en muy poco tiempo para poder encontrar las que tienen más probabilidades de tener éxito y no perder ninguna oportunidad. Esa es la promesa de pretotipar



Primer Director de Ingeniería de Google y empresario en serie **Alberto Savoia** , junto con los colaboradores **Richard Cox Braden** y **Leslie Barry** , tienen una formación desarrollada, herramientas e infraestructura para enseñar a cualquier organización cómo practicar pretotyping. Con su ayuda, aprenderá a probar todas sus ideas con una eficiencia sin precedentes para que su capacidad de innovar aumente exponencialmente.

VIDEO 2. LAS VENTAJAS DE CREAR UN PROTOTIPO DE UN SISTEMA

1. RECONOCIMIENTO
- conocer el problema
- simplificación y segmentación

2. EVOLUCIÓN
- Alcances
- Comportamiento de la aplicación

3. MODELADO
- Diseño del prototipo
- Navegación y operación

4. Especificación
- Alcances y navegación
- Apariencia

5. Revisión
- Modificaciones
- Opinión de todas las partes involucradas

Division de roles	
reconocimiento	analista/cliente
evaluacion	analaista
modelado	especialista UX
especificacion	especialista UX/analista
revision	analista/cliente

PROGRAMADOR	Conocimiento enteramente tecnico
	Empirico
	Integrador
Desarrollador	Conocimiento profunfo de procesos
	Profesional
	Creador
Demo	Hardcoded
	Impactante
	Sin escalabilidad
Prototipo	Sin diseño visual
	Explicativo
	Inductivo

Objetivos de un prototipo

- Análisis exitoso
- Pruebas de concepto
- Diseño de exploración

Metas máximas de un prototipo

- Evitar modificaciones posteriores
- Alcances NO modificables
- Tiempos exactos

Cliente como observador

- Nula participación
- Nula intervención

Garrido de la Cruz Karla Gabriela

Actividades semana 12 (Dic 7-11, 2020)

Video1. Free software, free society: Richard Stallman at TEDxGeneva 2014

Se considera el software libre como la primera batalla en la liberación de ciberespacio.

Una computadora es una maquina universal que ejecuta cualquier cosa que se pida porque cuenta con un programa que lo entienda y lo ejecute.

La computadora solo sabe recibir instrucciones y ejecutarla, recibir otra instrucción y ejecutarla

Al escribir el programa adecuado puedes conseguir cualquier cosa, o al menos casi todo.

Un grupo de usuarios pueden controlar un programa o este controla ellos.

Para que los usuarios puedan controlar el programa ellos necesitan 4 libertades esenciales

El software libre respeta la libertad de los usuarios y la comunidad

1. Estudiar el código fuente del programa y cambiarlo para que así ejecute la orden que quieras.

Código fuente

Normalmente cada programa tiene dos formas, hay una forma que puedes leer y entender si conoces el lenguaje de programación, este es el código fuente, es lo que los programadores escriben y modifica, y tenemos el ejecutable, es un conjunto de números que incluso un programador no puede entender.

Cada usuario puede realizar una copia, editarlo y ejecútalo, esto es un control individual.

2. Distribuir copias exactas

Se hacen copias y se redistribuyen o se venden las veces que quieras y como quieras.

3. Distribuir copias exactas, pero para las versiones modificadas

Se hacen copias y se redistribuyen o se venden las veces que quieras y como quieras. Si realmente tienes estas libertades el software es libre, los usuarios controlan el programa.

Software propietario:

A veces el programa fisgonea el usuario, a veces rastrea el usuario, a veces restringe al usuario e impide que los usuarios hagan lo que desean hacer, a veces el software borra los libros remotamente, a veces el desarrollador fuerza al usuario a instalar una actualización perjudicial amenazando con eliminar alguna otra funcionalidad.

Pueden incluso cambiar por la fuerza el software a distancia.

¿Cómo dejar de ser víctima?

Antiguamente se tenía que dejar de trabajar con equipo de cómputo, pro ya no, en 1983 se anuncia desarrollo un sistema operativo completamente libre llamado GNU, EN 1992 se tenía casi terminado.

Mantener la libertad requiere un sacrificio

1. Querer aplicaciones que no son del todo libres, si quieres libertad tienes que serlo sin esas aplicaciones.
2. Algunos sitios web ejecutan en el navegador software propietario escritos en JavaScript, sino quieres ejecutarlo como software propietario deberías instalar libreJS, que bloquea y envía el JavaScript no libre.
3. A veces los servidores te ofrecen realizar tus tareas “envíanos toda tu información, el servidor hace la tarea y le devuelve los resultados

Una gran parte de los servidores del mundo se ejecutan sobre GNU/Linux y otro software libre.

GARRIDO DE LA CRUZ KARLA GABRIELA

Actividades semana 13 (Dic 14-18, 2020)

Video 1. A Philosophy of Software Design | John Ousterhout

libro fue escrito por alguien que ha estado escribiendo código durante décadas, la prensa universitaria es la prensa de Stanford, y el libro que cubre las lecciones aprendidas durante la primera clase de diseño de software en Stanford midió mi interés. Aún así, me preguntaba cuánto aprendería sobre el diseño de software de la

experiencia parcialmente destilada de un aula, incluso si fuera un aula de Stanford. Mucho, como resultaría.

La mayor limitación en la escritura de software es nuestra capacidad para comprender los sistemas que estamos creando. (...) Cuanto más grande es el programa y más gente trabaja en él, más difícil es gestionar la complejidad. Las buenas herramientas de desarrollo pueden ayudarnos a lidiar con la complejidad. (...) Pero hay un límite a lo que podemos hacer solo con herramientas. (...) Los diseños más simples nos permiten construir sistemas más grandes y poderosos antes de que la complejidad se vuelva abrumadora.

odificación táctica versus estratégica : decidir cuándo invertir a largo plazo no es fácil. Tenga en cuenta si está apagando un incendio o si está construyendo a largo plazo. Esta observación coincide con mi experiencia: "piratear" algo juntos rápidamente, solo para que funcione, frente a la plataforma (hacerlo reutilizable, extensible) es una compensación difícil de hacer. Si bien este consejo suena simple, es el que requiere experiencia y tiempos de quemado para hacerlo bien. El desafío aquí es similar al dilema del desarrollo de software de moverse rápido, sin romper cosas .

Diseñar las cosas dos veces (Capítulo 11) es una sugerencia que llega cerca de casa. Este es un consejo que he estado sugiriendo a la gente para mejorar en el diseño de sistemas , mucho antes de leer este libro.

Los módulos profundos vs superficiales y el uso inteligente de capas (Capítulo 4 y 7) son capítulos, donde John observa cómo las abstracciones que tienen interfaces simples (módulos profundos) pero ocultan una funcionalidad compleja ayudan a reducir la complejidad de los programas. Lo hacen mejor que los módulos superficiales: módulos que tienen una implementación simple, pero interfaces complejas. Esto es algo en lo que no había pensado mucho antes, pero ciertamente suena cierto. La profundidad de un módulo es un concepto que antes no pensaba utilizar, pero lo estoy agregando a mi conjunto de herramientas. Gran parte del libro se basa en este concepto de profundidad del módulo:

Es más importante que un módulo tenga una interfaz simple que una implementación simple, es un pensamiento introducido en el Capítulo 8. Si bien mi experiencia lo confirma, especialmente cuando se habla de sistemas distribuidos y micro servicios. Encontré lógica y ejemplos que John solía obtener aquí como interesantes. Para la teoría, se basa en la lógica de que los módulos profundos fomentan la ocultación de

información y reducen la complejidad. Para la práctica, toma ejemplos de estudiantes que implementan una clase que administra archivos para una tarea de editor de texto GUI. Los estudiantes que eligieron una implementación simple, pero interfaces complejas, exponiendo el concepto de líneas, se encontraron con muchos más problemas y complejidad. Lucharon más con esta interfaz que aquellos que optaron por una interfaz simple, con una implementación compleja: una interfaz basada en caracteres.

Las capas deben eliminar, no agregar complejidad a un sistema (Capítulo 7). El libro se refiere a esto como "capa diferente, abstracción diferente", argumentando en contra del uso de métodos de llamada directa o clases decorativas. Si bien estoy de acuerdo en que las capas de transferencia agregan complejidad al sistema y es mejor evitarlas, mi experiencia cuando trabajo en sistemas de producción me dice que no es tan simple. Trabajar dentro de una aplicación o base de código, esta complejidad es más trivial de detectar y eliminar. Sin embargo, cuando se trabaja con servicios independientes, es un proceso más laborioso. Es un buen recordatorio tanto para resistir la tentación de construir servicios que envuelvan demasiado, como para realizar un seguimiento de este tipo de deuda arquitectónica.

El ocultamiento y la filtración de información (Capítulo 5) es otra visión de lo que es una buena abstracción y cómo deben comportarse las interfaces eficientes. Aquellos que han diseñado API probablemente hayan tenido experiencia de primera mano sobre por qué la filtración de información más allá del mínimo básico conduce a una deuda de tecnología y arquitectura más adelante. John también menciona un ejemplo evidente de mal diseño de API: exponer en exceso los componentes internos.

La fuga de información correlacionada con clases superficiales fue una observación interesante: la primera de varias basadas en el libro, no basadas en la propia experiencia de John, sino en su análisis del trabajo del estudiante. Se dio cuenta de cómo los estudiantes que dividían su código en muchas clases pequeñas, poco profundas, terminaban con mucha lógica duplicada, causada por la filtración de información.

La generalización conduce a una mejor ocultación de la información. Los módulos de uso general / reutilizables son más profundos: tienen una lógica interna más compleja, pero interfaces más simples. No sorprende a nadie que haya intentado crear API más genéricas. Sin embargo, llevando el argumento anterior más allá, también significa que la generalidad conduce a un mejor ocultamiento de la información. ¿Quiere tener una arquitectura más sencilla? Considere la posibilidad de generalizar componentes. El ejemplo de la construcción de un editor de texto durante el semestre se utiliza para probar este punto, con ejemplos.

Las compensaciones al combinar o separar implementaciones dentro de módulos o interfaces (Capítulo 9) es un debate interesante del que no conozco una única mejor respuesta. Compartimos esta opinión con John, quien también está de acuerdo en que la mejor solución dependerá del contexto. John recopila algunas reglas prácticas que sugiere usar al decidir combinar o separar. Combine cuando logre una interfaz más simple, para reducir la duplicación o cuando las implementaciones compartan datos / información. Código separado de propósito general y de propósito especial.

Si bien todo lo anterior es sensato, personalmente todavía prefiero las interfaces de un solo propósito, incluso si eso podría dejar alguna implementación por separado, que técnicamente podría combinarse. En el caso de los micro servicios, un principio rector importante es evitar separar servicios mientras usan / modifican la misma fuente de datos. Un servicio más grande que es cohesivo resulta en menos complejidad que varios servicios más pequeños, manipulando la misma fuente de datos.

La importancia de una nomenclatura buena y simple (Capítulo 14) refleja mi experiencia sobre cómo los nombres simples a menudo significan funcionalidad simple. Los nombres complejos, o la dificultad para nombrar algo, suelen ser un código o un olor a arquitectura. John menciona que la nomenclatura constante contribuye a una menor complejidad, algo con lo que estoy totalmente de acuerdo.

Video 2. Proyectos de calidad comienzan con requisitos de calidad

Los proyectos de software de calidad requieren de una especificación de requisitos de calidad. Diversos estudios muestran que buena parte de los defectos que se presentan en un software, se originan en la especificación de requisitos. Y lo más importante, entre más temprano se detecta un problema en un proyecto, más económica es su corrección. Por tanto, el primer paso en dirección a la mejora de calidad de un software es la mejora de calidad de la especificación de requisitos. El propósito de webinar es aborda en lo siguiente:

-¿Qué es requisito de software?

1. Condición o capacidad que un usuario necesita para resolver un problema o lograr un objetivo

2. Condición o capacidad que debe cumplir o poseer un sistema o uno de sus componentes para satisfacer un contrato, estándar, especificación u otra documentación formalmente impuesta

3. Representación documentada de una condición o capacidad como en los anteriores puntos

-¿Qué es especificación de requisitos?

Es un conjunto de requisitos que:

· Ayuda a los clientes a describir con precisión lo que se desean obtener (de un software)

· Ayuda a los desarrolladores a entender exactamente lo que quiere el cliente

-El papel de la especificación de requisitos en un proyecto de software.

· Ser un contrato entre cliente y desarrolladores

1. No se debe enfocar en aspectos de diseño o implementación

2. Además de ser detallado, debe promover la comunicación entre las dos partes

· El nivel de confianza entre las dos partes determina el nivel de detalle.

-¿Cuál es el nivel de detalle apropiado para la especificación?

-¿Cuáles son los criterios de calidad deseados?

Correcta: cada requisito satisface la necesidad o demanda legitima del negocio. Es decir, debe trazar alguna necesidad (requisito de negocio)

· No contiene requisitos superfluos

· Menos riesgos de scope creep y gold plating

Clara (sin ambigüedad): Tiene una interpretación única para todo el público. El lenguaje natural, casi siempre usado para describir requisitos, es inherentemente ambiguo. Tips:

ü Utilice las mejores prácticas para la estructura de texto

ü Use un glosario para definir términos del contexto

ü No use termino subjetivos o indeterminados: “etc.”, “entre otros”, “bueno”, “amigable”, “flexible”, “portable”, “razonable”, “intuitivo”

ü Use ejemplos

Completa: Todos los elementos relevantes del contexto de interés (el dominio del problema) están descritos. Ejemplos:

Ø Funcionalidades, aspectos de calidad, restricciones de diseño, interfaces externas

Ø Definición de todas las respuestas para cada tipo posible de entrada al software

Ø Rótulos y referencias a todas las figuras, tablas y diagramas en la especificación

· La trazabilidad ayuda a garantizar completitud

· Especificación con partes “para definir” es incompleta

Consistente: No existen contradicciones entre los documentos de requisitos, sea en un mismo nivel o en niveles diferentes.

Ejemplos de contradicciones:

Ø Temporalidad: REQ-03 indica que el evento A predice al evento B; REQ-12 indica que los eventos A y B son simultáneos

Ø Dos requisitos utilizan diferentes nombres para el mismo objeto del mundo real

A menudo, la inconsistencia surge de solicitudes de cambios mas asimilados en la especificación

Modificable: Pueden realizarse cambios de una manera fácil, completa y consistente, sin comprometer la estructura y estilo de la especificación. Directrices:

Ø Tener una organización coherente y fácil de usar que incluye, por ejemplo, una tabla de contenido, un índice, glosario y control de cambios

Ø No ser redundante

Ø Expresar cada requisito por separado en lugar de combinarlo con otros requisitos

Priorizada: Cada requisito de la especificación tiene atribuido un valor de importancia relativa basado en uno o más criterios, por ej: el riesgo, costo

Ø Una buena priorización asegura que el esfuerzo se centrara en las necesidades mas criticas, lo que reduce los riesgos del proyecto

La priorización permite:

Ø Determinar los requisitos que se deben analizar primero

Ø Planificar que requisitos implementar primero

Ø Cuanto tiempo o atención se destinará a los requisitos

Verificable: Algún método (costo-beneficio aceptable) pueden ser establecido para demostrar objetivamente que le software cumple con el requisito. De lo contrario, este debe ser eliminado o revisado. Ejemplos:

Ø El requisito no funcional “la interfaz de usuario debe ser fácil de usar” no es verificable debido a su subjetividad

Ø El requisito no funciona “95% de las operaciones de procesar en menos de 1 segundo cada uno” es verificable porque se expresa en términos mensurables

Trazable: Establece la relación entre los requisitos, sus fuentes y sus productos derivados

Hace la especificación mas modificable facilita el análisis del impacto de los cambios, y verificación de su corrección y completitud

Ø Ayuda a garantizar el cumplimiento del software con sus requisitos, identificando los requisitos que faltan o sobran y verifican si todos los objetivos de negocio están cubiertos por los requisitos y productos

Ø Ayuda en la gestión de Riesgos: requisitos con muchas relaciones tienen mayores riesgos.

Verificación de requisitos: Asegura que las especificaciones y modelos cumplen con los estándares. Ej. Errores de notación, datos incompletos o inconsistentes, malas prácticas.

Técnicas principales: Listas de verificación y revisiones

Ø Constituye una comprobación fina para asegurarse de que los requisitos están listos para la validación por los clientes.

Ø Validación de requisitos: Asegura que todos los requisitos estén alineados con os requisitos del negocio. Es decir, tratan de garantizar que se cumplen todas las necesidades de negocio del cliente

Ø Técnicas principales: Revisiones y prototipos

conclusiones
Requisitos son claves para que los proyectos logren éxito
Requisitos no equivalen a documentación
Especificación detallada no siempre significa mejor calidad
Verificación y validación deben siempre estar presentes
Proyectos con calidad comienzan con requisitos de calidad

GARRIDO DE LA CRUZ KARLA GABRIELA

Actividades semana 14 (Ene 7-8, 2021)

VIDEO. 1 The unexpected benefit of celebrating failure | Astro Teller

Emprendedor, inventor, autor y director de X (antes Google X), Astro Teller es un hombre con una misión con ideas que podrían remodelar nuestras vidas y ayudar a hacer del mundo un lugar maravilloso a través de la tecnología. En esta increíble charla TED, nos da una mirada privilegiada a lo que se llama la "fábrica de disparos a la luna", donde él y su equipo están trabajando para encontrar soluciones a algunos de los problemas más grandes del mundo. "Los grandes sueños no son solo visiones", dice Astro Teller, "son visiones unidas a estrategias para hacerlos realidad". Teller es propenso a crear una organización en la que las personas se sientan cómodas trabajando en proyectos grandes y arriesgados mientras están completamente preparadas para enfrentar (e incluso celebrar) el fracaso .

Se aseguró de que se pusiera en marcha un plan para hacerlo si era posible. Así de grandes son los sueños. Los grandes sueños no son solo visiones, son visiones unidas a estrategias para hacerlos realidad. Tengo la increíble suerte de trabajar en una fábrica de moonshot.

Estos inventores, ingenieros y fabricantes están ideando tecnologías que esperamos puedan hacer del mundo un lugar maravilloso. Usamos la palabra «moonshots» para recordarnos que debemos mantener nuestras visiones en grande, para seguir soñando. Y usamos la palabra «fábrica» para recordarnos a nosotros mismos que queremos tener visiones concretas, planes concretos para hacerlas realidad. Aquí está nuestro plano de disparo a la luna.

La fábrica de disparos a la luna es un lugar desordenado. Tenemos este equilibrio interesante en el que permitimos que nuestro optimismo desenfrenado alimente nuestras visiones. Pero también aprovechamos el escepticismo entusiasta para dar vida, dar realidad a esas visiones. El año pasado matamos un proyecto en agricultura vertical automatizada.

Una de cada nueve personas en el mundo sufre de desnutrición. Así que esto es un tiro a la luna que debe suceder. La agricultura vertical utiliza 10 veces menos agua y cien veces menos tierra que la agricultura convencional. Y como puede cultivar la comida cerca de donde se consume, no tiene que transportarla a grandes distancias.

Así que matamos el proyecto. Pagamos enormes costos en recursos y daños ambientales para enviar mercancías a todo el mundo. Pero por muy baratos que hubieran sido en volumen, resultó que costaría cerca de $ 200 millones diseñar y construir el primero. $ 200 millones es demasiado caro.

Debido a que X está estructurado con estos ciclos de retroalimentación ajustados de cometer errores y aprender y nuevos diseños, no podemos gastar $ 200 millones para obtener el primer punto de datos sobre si estamos en el camino correcto o no. Así que también matamos este proyecto. Descubrir una falla importante en un proyecto no siempre significa que termina el proyecto. A veces, realmente nos lleva a un camino más productivo.

Con 1,2 millones de personas que mueren en las carreteras de todo el mundo cada año, construir un automóvil que se conduzca solo fue una oportunidad natural. Y lo que descubrimos fue que nuestro plan de que los autos hicieran casi todo el manejo y solo entregárselos a los usuarios en caso de emergencia era un plan realmente malo.

VIDEO 2. Retos y soluciones de trabajar con Requerimientos de Software

La disciplina de ingeniería de software que consiste en un uso sistemático y repetitivo de técnicas que abarcan las actividades de IDENTIFICACION, DOCUMENTACION Y MANTENIMIENTO DE UN CONJUNO DE REQUERIMIENTOS para el software, con el fin de que estos cumplas con los objetivos de negocio y sean de calidad

47% de los fracasos en proyectos se deben a la gestión deficiente de los requerimientos

20% de los defectos tienen su origen en requerimientos

Encontrar y corregir defectos en el software después de entregarlo es >10 x más costoso que hacerlo durante el trabajo de requerimientos

Ø El requerimiento de software es la condición o la capacidad que un usuario necesita para resolver un problema o lograr un objetivo

Ø Condición o capacidad que debe cumplir o poseer un sistema o uno de sus componentes para satisfacer un contrato, estándar, especificación u otra documentación formalmente impuesta

Ø Representación documentada de una condición o capacidad como en los pasos anteriores

COMUNICACIÓN

Proporcionar información sin subjetividad o ambigüedad: Los requerimientos se expresan a menudo en lenguaje natural, lo que facilita la comprensión, pero da lugar a varias interpretaciones

Falla en la interpretación del mensaje entre el analista de requerimientos e interesados. La propagación errónea de los requerimientos para otros miembros del equipo involucrados en el proyecto.

Reto: ACCESO A LOS INTERESADOS

Ø No siempre está al alcance del analista seleccionar las personas con quien levantar requerimientos

Ø A veces un intermediario es seleccionado para desempeñar el rol de un interesado. Esto es común cuando el interesado es externo a la organización (cliente, proveedor, aliado, etc.)

Ø Esto significa un riesgo considerable de que la información recolectada no sea la mas adecuada

Si la dificultad es la falta de autoridad para elegir a los interesados, involucrar al director de proyectos es fundamental para una solución

Otra alternativa es buscar personas adicionales que también puedan tener la información deseada, u otras fuentes de información, por ejemplo: documentación, existente, observación.

Ø Esta dificultad varía desde aquellos que no saben decir lo que quieren, hasta aquellos que dicen la necesidad incorrecta. Y esto es un escenario muy frecuente

Conflictos

Ø Los conflictos aumentan en la proporción de la cantidad de interesados, ejemplo

Ø Solicitudes de distintos interesados que no se pueden cumplir simultáneamente, datos no consistentes del proceso de negocio

Ø Solicitudes fuera del alcance del proyecto

Ø Interesados enemigos entre si

Ø Falla de sintonía entre las áreas de negocio

GARRIDO DE LA CRUZ KARLA GABRIELA

Actividades semana 15 (Ene 11-15, 2021)

VIDEO 1. SPRINT | Jake Knapp & John Zeratsky | Talks at Google

Diseñar prototipos y validarlos rápidamente. Es un resumen del libro “Sprint: Cómo resolver grandes problemas y testear nuevas ideas en sólo 5 días”. Sprint es un DIY (Do It Yourself) para aprender a hacer sprints de una semana y verificar la viabilidad de una idea desde el punto de vista del negocio. En tan sólo 5 días aprendes a convertirte en una máquina de resolver problemas.

Se trata de un magnífico libro escrito por Jake Knapp, John Zeratsky y Braden Kowitzy, que son tres genios de Google Ventures (GV).

El método que se explica en el libro fue diseñado poco a poco, ayudando a crear o validar productos como Gmail o Hangouts para luego convertirse en el proceso de diseño que se aprende en Google Ventures (GV). No tienes más que darte una vuelta por el porfolio de startups financiadas por GV para ver que las cosas funcionan bien en ese fondo: la gente termina por montar empresas de verdad tipo Slack, Uber, Medium o Hubspot, ¡entre otras!

“The bigger the challenge, the better the sprint- Cuanto mayor sea el desafío, mejor será el sprint.”

Para validar la percepción del cliente antes de construir el producto o implementar el servicio, y esto tiene mucho que ver con Lean, Design Thinking y los métodos de desarrollo Agile, ¿verdad? La diferencia entre Sprint y estas metodologías es que el primero es un método cerrado, una propuesta completa en sí misma. Si lo aplicas bien, en 5 días se puede obtener la validación de prototipo concreto y conciso.

Todas ellas metodologías promueven la interacción directa con el usuario/cliente para aprender de su experiencia y dar solución a las necesidades reales del mismo.

Sprint ayuda a enfrentarse con los retos:

Si tu reto implica una gran inversión de tiempo y dinero, este método te ayuda a establecer la dirección adecuada antes de gastarlos.

Si lo que te falta es tiempo para implementar un proyecto, necesitas tener una buena solución rápidamente, Sprint aporta velocidad y te ayudará.

Si simplemente estás atascado con un reto y no sabes por dónde tirar, no lo dudes e intenta desbloquearte abordando un sprint de una semana a ver qué pasa.

VIDEO 2. Introducción a los patrones de diseño

Un patrón de diseño es, la solución a un problema de diseño, el cual debe haber comprobado su efectividad resolviendo problemas similares en el pasado, también tiene que ser reutilizable, por lo que se deben poder usar para resolver problemas parecidos en contextos diferentes.

Importancia de los PDD

Ø Demuestra la madurez de un programador software

Ø Evita reinventar la rueda

Ø Agiliza el desarrollo de software

Ø Se basa en las mejores prácticas de programa

Ø Permite utilizar un vocabulario común

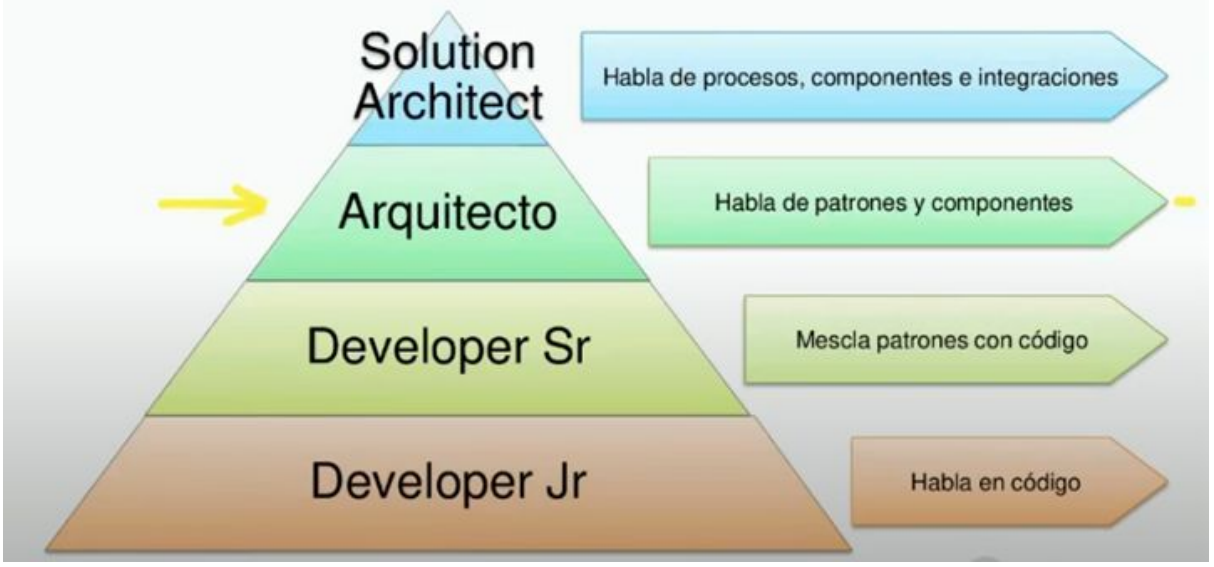
Lo que se busca con los PDD

Ø Proporcionar un catálogo de soluciones probadas de diseño para problemas comunes conocidos.

Ø Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente

Ø Crear un lenguaje estándar entre los desarrolladores

Ø Facilitar el aprendizaje a nuevas generaciones de programadores


Solution Architect
Habla de procesos, componentes e integraciones
Arquitecto
Habla de patrones y componentes
Developer Sr
Mescla patrones con código
Developer Jr
Habla en código