Resumen

Teoría

Indice:

UNIDAD 1

PARCIAL 1

Introducción a la Ingeniería del Software

Que es Software?

Programas de cómputo y documentación asociada.
Es todo lo que se genera como producto de las actividades del proceso de desarrollo de software (Requerimientos, analisis, implementacion, diseño, pruebas, despliegue).

Que es la Ingenieria de Software?

La ingeniería de software es una disciplina de la ingeniería que se interesa por todos los aspectos de la producción de software, desde las primeras etapas de la especificacion del sistema hasta el mantenimiento del sistema despues de que se pone en ejecucion.

Disciplinas que conforman la ingeniería de software

- Disciplinas técnicas: nos referimos a actividades que aportan al desarrollo de software como producto. Ellas son:
    - Toma de requerimientos
    - Análisis de requerimiento
    - Diseño de software
    - Implementación de software
    - Prueba de software
    - Despliegue de un software
    - Documentación de software
    - Capacitaciones a usuarios
- Disciplinas de gestión: hacen referencia a actividades de planificación, monitoreo y control del proyecto de desarrollo de software. Se utilizan, por ejemplo, metodologías LEAN Agile de gestión de proyectos.
- Disciplinas de soporte: son disciplinas transversales a los procesos de software, que permiten verificar la integridad y calidad de un producto de software. Entre ellas se incluyen:
    - Gestión de Configuración del Software (SCM)
    - Toma de métricas
    - Aseguramiento de calidad (QA - Quality Assurance)

Ciclos de vida (Modelos de Proceso) y su influencia en la Administración de Proyectos de Software

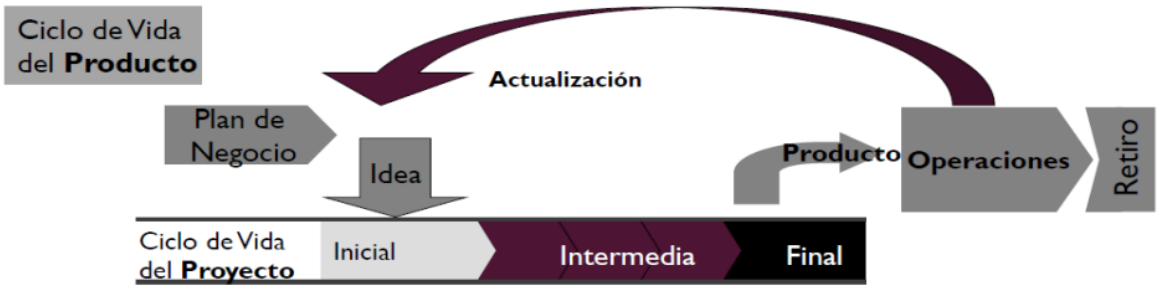
Que es un Ciclo de Vida?

Se denomina ciclo de vida a una serie de pasos a través de los cuales un producto o un proyecto progresa en el tiempo. Es decir, que es una representación simplificada de un proceso, el cuál define elementos del proceso (actividades estructurales, productos del trabajo, tareas, etc.) y el flujo del proceso (o flujo de trabajo), que especifica la relación y el orden de dichos elementos. Es decir, son modelos genéricos de los procesos de software; y establecen los criterios que utiliza para determinar si se debe pasar de una tarea a la siguiente. Y son independientes de los procedimientos de cada actividad del ciclo de vida.

Relación entre el ciclo de vida del proyecto y del producto

No existe un impacto entre los ciclos de vida de cada uno, sino que el ciclo de vida del producto siempre es mayor que el ciclo de vida del proyecto, ya que el ciclo de vida del proyecto dura lo que dura el desarrollo del software, en cada una de sus versiones. En cambio, el ciclo de vida del producto dura hasta que el software se deje de utilizar, dependiendo esto de la madurez que tenga el producto en base a las necesidades del mercado.

Es posible que un producto tenga varios proyectos en su ciclo de vida, debido a, constantes cambios y/o actualizaciones que se vayan realizando. Por lo que, dentro del ciclo de vida del producto, se pueden desarrollar varios ciclos de vida de proyectos.



Clasificacion de los Ciclos de Vida

Modelo Secuencial

Este modelo dispone de las actividades de forma lineal, es decir, que el proyecto progresa a través de una secuencia ordenada de pasos (fases) y una actividad no puede iniciar sin que la precedente haya sido finalizada. El avance entre las fases se da tras una revisión al final de cada una de estas para determinar si el software está listo para avanzar.

Se suele utilizar en aquellos sistemas donde los requerimientos se comprenden bien y son exhaustivos ya que éstos se congelan al finalizar la etapa de toma de requerimientos.

Generalmente, este tipo de modelos está dirigido por documentos, es decir, el trabajo principal del producto es la documentación del software entre las distintas fases.

Modelos Iterativo / Incremental

Este modelo aplica sucesivas iteraciones en forma escalonada a medida que avanza el calendario de actividades. Cada iteración produce un incremento de software funcional potencialmente entregable. Usualmente los primeros incrementos incluyen las funciones básicas/críticas que más requiere el cliente. Este enfoque vincula las actividades de especificación, desarrollo y validación. El sistema se desarrolla como una serie de versiones (incrementos) y cada una añade funcionalidades a la versión anterior. Este ciclo de vida se utiliza en metodologías ágiles.

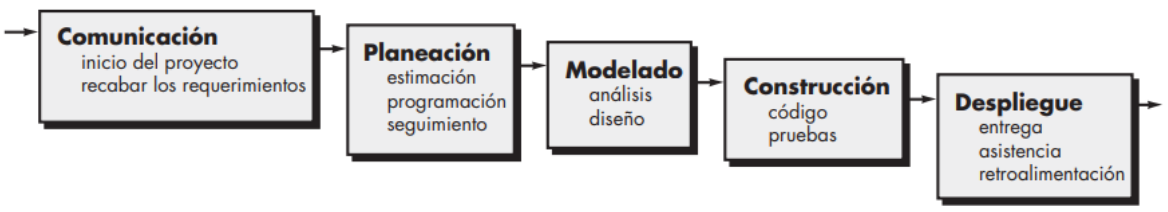
Modelo Recursivo

El modelo recursivo es utilizado para gestionar los riesgos del desarrollo de sistemas complejos a gran escala. Requiere de la intervención del cliente.

Ciclo de Vida	Procesos de desarrollo que lo admite	Características que lo hacen elegible
Secuencial	Definidos	• Alta certeza (baja incertidumbre). • La organización tiene una forma tradicional de trabajar. • No se pueden realizar entregas parciales del software (se debe entregar todo junto). • Eliminar riesgos de planificación.
Iterativo/incremental	Definidos o Empíricos	• Existe incertidumbre. • Volatilidad de requerimientos. • Posibilidad de entregas parciales. • Uso en etapas tempranas del producto.
Recursivo	Definidos	• Mucho control de riesgos en cada iteración. • No se pueden realizar entregas parciales.

Modelos de Ciclos de Vida

Modelo de Proceso en Cascada

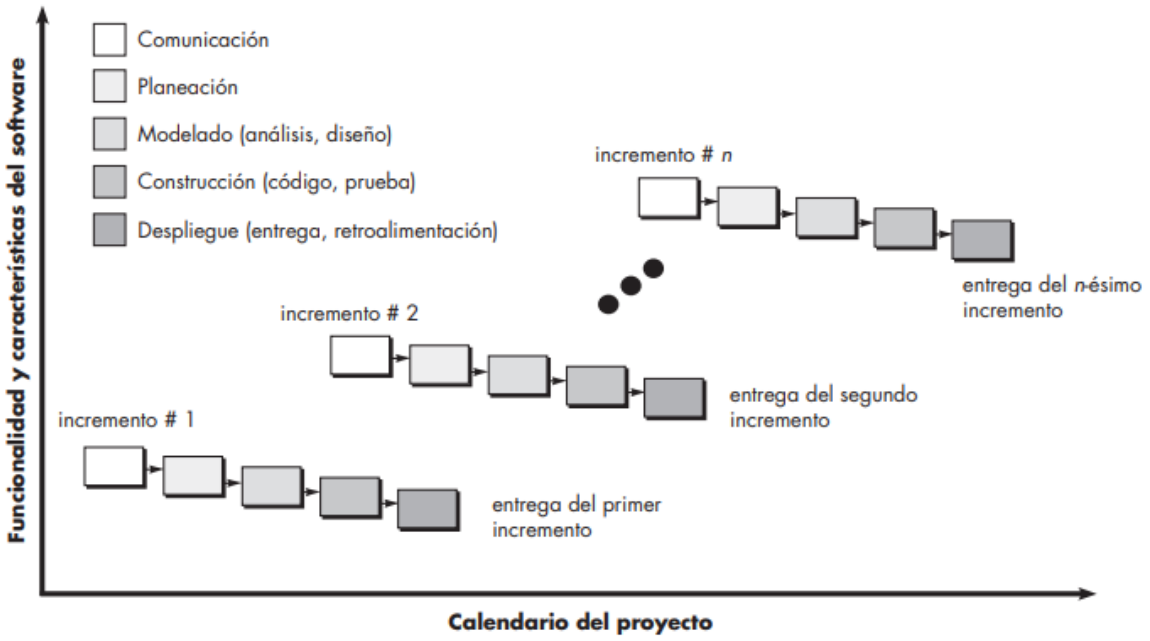


Tambien llamado Ciclo de Vida Clasico, sugiere un enfoque sistematico y secuencial para el desarrollo del software.

Es el paradigma mas antiguo de la ingenieria de software, sin embargo posee grandes problemas:

- Es raro que los proyectos reales signa el flujo secuencial propuesto por el modelo.
- Es dificil para el cliente enunciar en forma explicita todos los requerimientos, y este modelo precisa que se lo haga.
- El cliente debe tener paciencia, ya que no se dispondra de una version funcional del programa hasta que el proyecto este avanzado.

Modelo de Proceso Incremental



Este modelo aplica secuencias lineales en forma escalonada a medida que avanza el calendario de actividades. Cada secuencia produce **INCREMENTOS** de software.

En este modelo es normal que con el primer incremento se obtenga el producto fundamental, y con los siguientes incrementos se obtenga un producto mas detallado.

> 💡 **ES IMPORTANTE OBSERVAR QUE TODOS LOS MODELOS DE PROCESO AGILES USAN LA FILOSOFIA INCREMENTAL.**

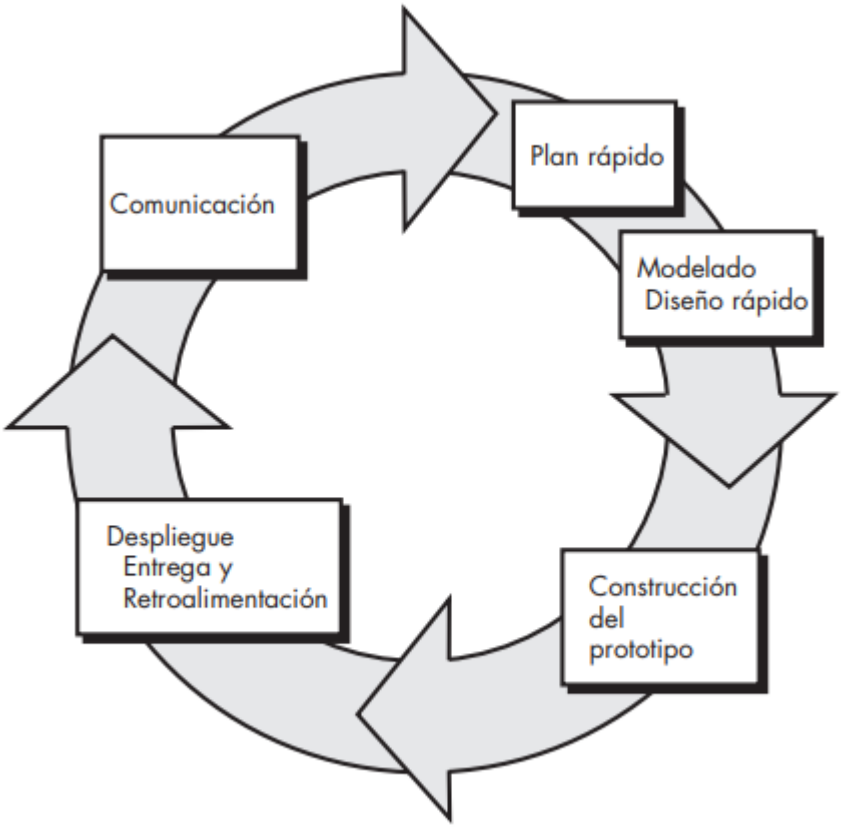
Modelo de Proceso Evolutivo

Este modelo se basa en que el software evoluciona con el tiempo. Es normal que los requerimientos del negocio y del producto cambien conforme avanza el desarrollo. Es por esto que este modelo busca **ADAPTARSE** a un producto que evoluciona con el tiempo.

Estos modelos son iterativos y se caracterizan por la manera en que permiten desarrollar versiones cada vez mas completas de software.

Dos modelos comunes evolutivos:

- Prototipos:
  El prototipo sirve como primer sistema, algunos se construyen para ser desechables y otros son evolutivos, es decir que se transforman poco a poco en el sistema real.



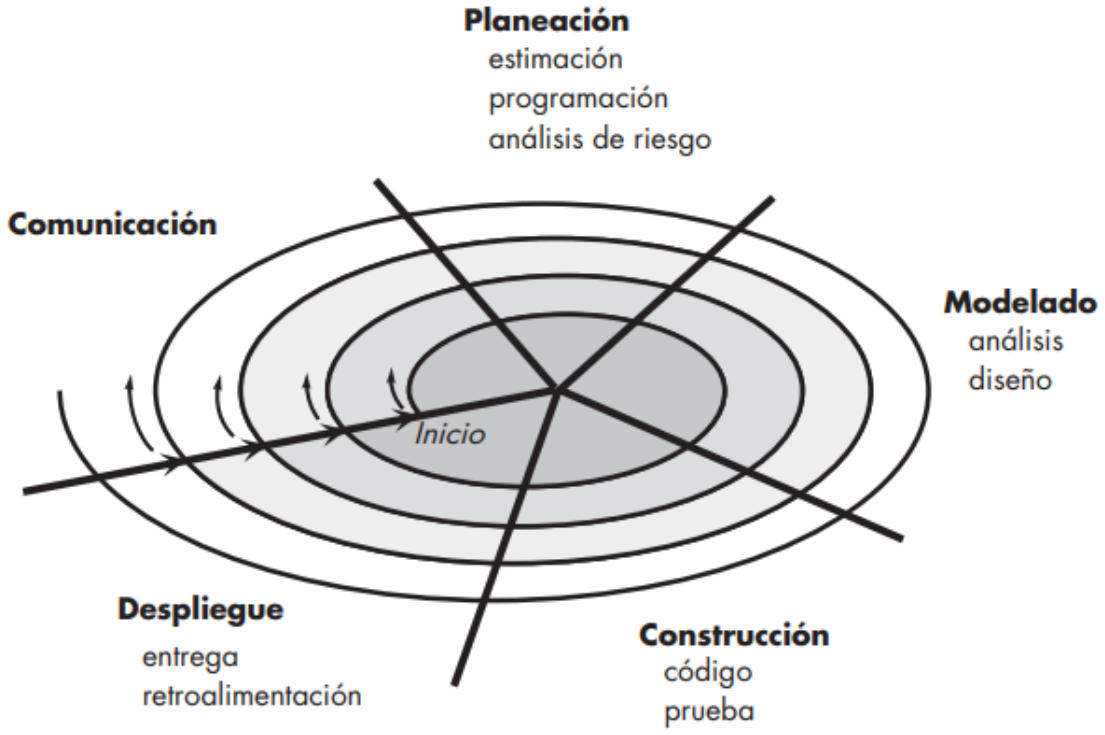
Desventajas:

- Los participantes pueden ver una version funcional del producto de software y no darse cuenta de que el prototipo se obtuvo de forma

caprichosa, pudiendo no tener en cuenta la calidad y la capacidad para darle mantenimiento a largo plazo.

    - Puede que con el fin de obtener el prototipo rapido, se utilicen sistemas operativos, lenguajes de programacion o algoritmos inapropiados, que en un primer momento se utilizaran por comodidad y facilidad pero que a futuro esas opciones sean inadecuadas.

- Espiral:
  Es un modelo que mezcla la naturaleza iterativa de hacer prototipos con los aspectos controlados y sistemicos del cascada.
  Con el empleo de este modelo, el software se desarrolla en una serie de entregas evolutivas, durante las primeras iteraciones lo que se entrega puede ser un modelo o prototipo y luego en las iteraciones posteriores se producen versiones cada vez mas complejas.
  Este modelo tiene muy en cuenta el
  Riesgo, este se tiene en cuenta en cada iteracion.

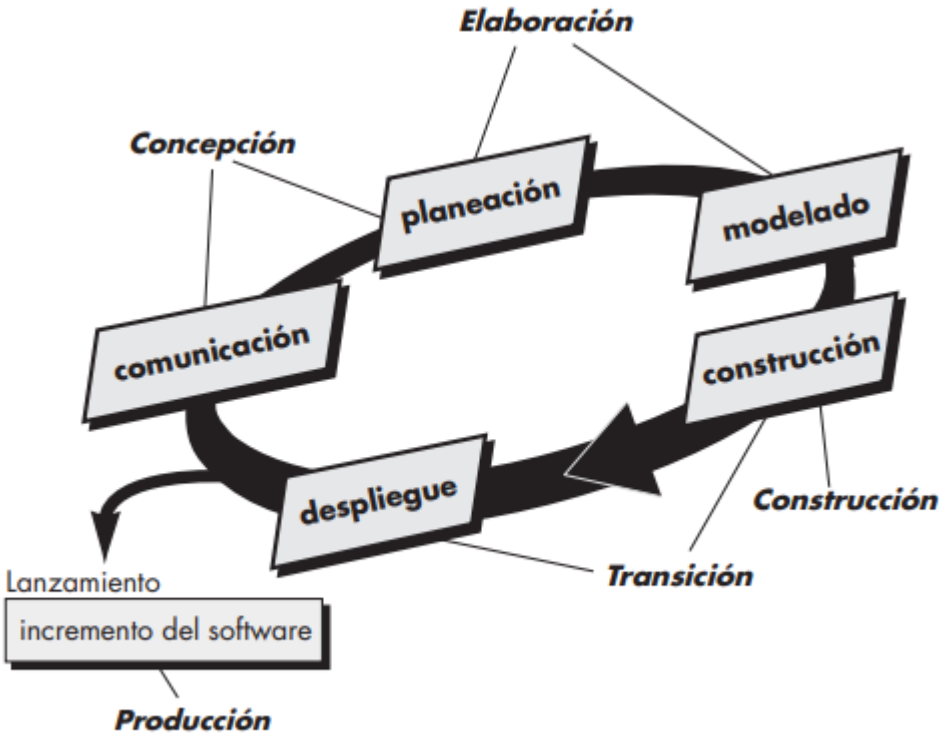


  Desventajas:

    - Demanda mucha experiencia en la evaluacion del riesgo y para este modelo el riesgo es muy importante.

- Puede parecer que este enfoque es incontrolable, lo que asusta a los clientes.

Proceso Unificado de Desarrollo



El proceso unificado sugiere un proceso iterativo e incremental.

El proceso de software debe ser:

- Impulsado por el caso de uso.
- Centrado en la arquitectura.
- Iterativo e Incremental.

Desarrollo Agil

Que es Agil?

Agil es una ideologia con un conjunto definido de principios que guian el desarrollo del producto.

⚠ NO ES UNA METODOLOGIA O PROCESO

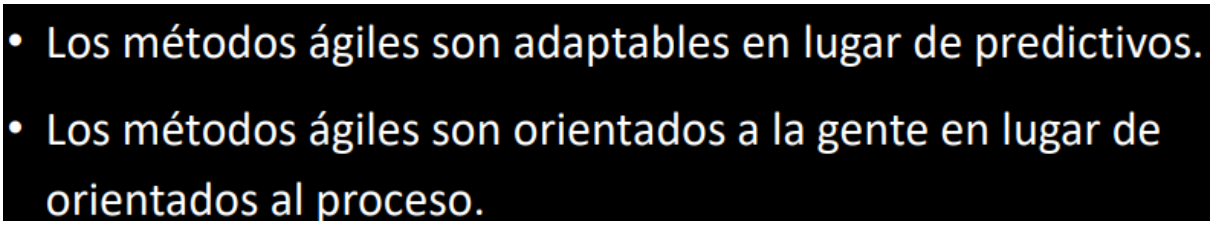
Es el balance entre ningun proceso y demasiado proceso.

Que es un Proceso / Metodo Agil?

Cualquier proceso de software agil se caracteriza por la forma en la que aborda cierto numero de suposiciones clave acerca de la mayoria de proyectos de software:

1. Es difícil predecir qué requerimientos de software persistirán y cuáles cambiarán. También es difícil pronosticar cómo cambiarán las prioridades del cliente a medida que avanza el proyecto.
2. Para muchos tipos de software el diseño y la construccion deben ejecutarse en forma simultanea.
3. El analisis, diseño, construccion y pruebas no son tan predecibles como nos gustaria (desde el punto de vista de la planeacion).

Para resolver esto los procesos agiles deben ser **ADAPTABLES.** Esta adaptacion debe ser incremental. Entonces deben entregarse incrementos de software en periodos cortos de tiempo, de forma que la adaptacion vaya a ritmo con el cambio.



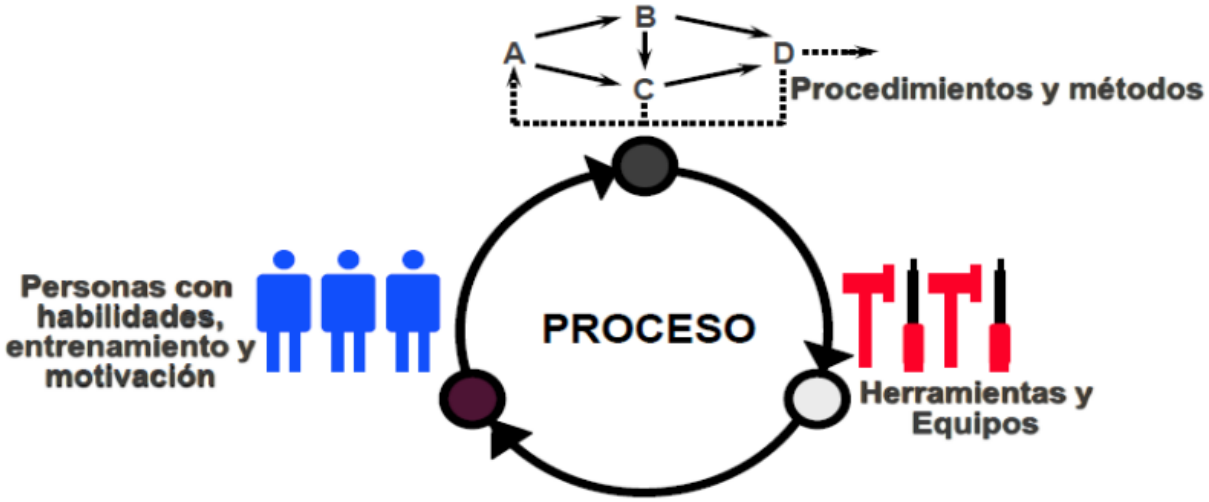
Procesos de Desarrollo Empíricos vs. Definidos

Que es un Proceso?

Conjunto de actividades ejecutadas para un proposito dado.

Existen 3 factores determinantes en un proceso:

- Procedimientos y Metodos: La adaptacion de los procesos establecidos para su implementacion, dependiendo del contexto es esencial para la calidad resultante. Es decir se deben definir y escribir los procedimientos.
- Personas Motivadas, Capacitadas y con Habilidades: De las personas depende el producto.
- Herramientas y Equipo: Materiales necesarios para llevar a cabo el proceso que nos permiten que las entradas se transformen en salidas.



Proceso Definido

Estos procesos son deterministas y predican que de las mismas entradas se obtienen las mismas salidas. Lo cual en realidad no es asi, por que no es posible obtener el mismo resultado con las mismas entradas porque estas dependen exclusivamente del contexto, momento y personas.

- Buscan dar una forma de trabajo institucionalizada para todos los equipos con intencion de lograr previsibilidad, este enfoque se basa en que los requerimientos son fijos y nunca cambian, lo cual no es correcto, siempre cambian.
  Por ejemplo: Si hay una persona en un area de la cual es responsable debe estar escrito y rodas las otras areas adaptarse a ese rol definido.
- Como ventaja se facilita el recambio de gente.
- Estos procesos pueden adoptar cualquier ciclo de vida.

Proceso Empirico

- Estan basados en experiencias.
- Se basan en el empoderamiento de los equipos los cuales estan Auto Gestionados.
  Estos equipos de trbaajos estan capacitados, motivados y poseen experiencia.
- La experiencia es propia y se obtiene por uno mismo. Por esto es que la misma no es extrapolable, es del equipo y solo le sirve para ese equipo.

- En los procesos empiricos se pueden usar solo ciclos de vida iterativos-incrementales.
- Los 3 pilares de los procesos empiricos son:
    - Transparencia: Se debe conseguir que la informacion relevante del proceso este disponible para todos y que todos puedan comprenderla.
    - Inspeccion: Se debe revisar y evaluar regularmente el proceso y los resultados para detectar problemas, oportunidades de mejora y garantizar que el equipo y el producto estén alineados con los objetivos.
    - Adaptacion: Son los ajustes que se realizan para mantener la meta alcanzable, es gracias a la transparencia e inspeccion que se pueden encontrar los puntos de mejora.

Componentes de un Proyecto de Sistemas de Información

Que es un Proyecto?

Es una unidad de gestion, organizacion temporal del trabajo. Gestiona personas y administra recursos.

Caracteristicas de un Proyecto

- Resultado: El resultado de un proyecto debe ser unico.
- Esfuerzo Temporal: Fecha de inicio y fin definida.
- Elaboracion gradual: Los objetivos se cumplen con objetivos mas pequeños que se realizan a lo largo del tiempo.
- Tareas Interrelacionadas basadas en esfuerzos y recursos: Las actividades estan relacionadas entre si y dependen de la asignacion adecuada de esfuerzo y recursos.

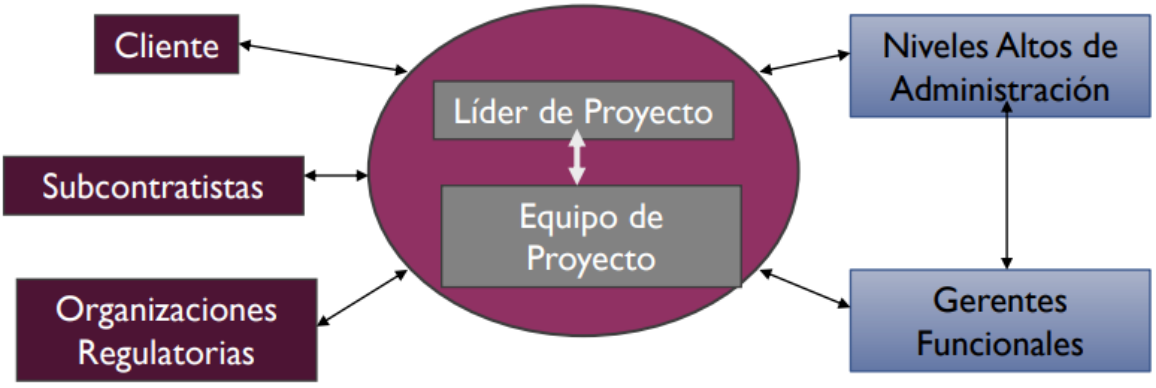
Que es la Administracion de Proyectos?

Es la aplicacion de conocimientos, habilidades, herramientas y tecnicas a las actividades del proyecto para satisfacer los requerimientos del proyecto.

Administrar un proyecto incluye:

- Identificar los requerimientos.

- Establecer objetivos claros y alcanzables.
- Adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados.



Que es un equipo de Proyecto?

Un grupo de personas comprometidas con alcanzar un conjunto de objetivos del cual son mutuamentes responsables.

Caracteristicas:

- Diversos conocimientos y habilidades.
- Posibilidad de trabajar juntos efectivamente.
- Sentido de responsabilidad como unidad.

Artefacto Principal de un Proyecto:

PLAN DE PROYECTO

Es un documento donde se define:

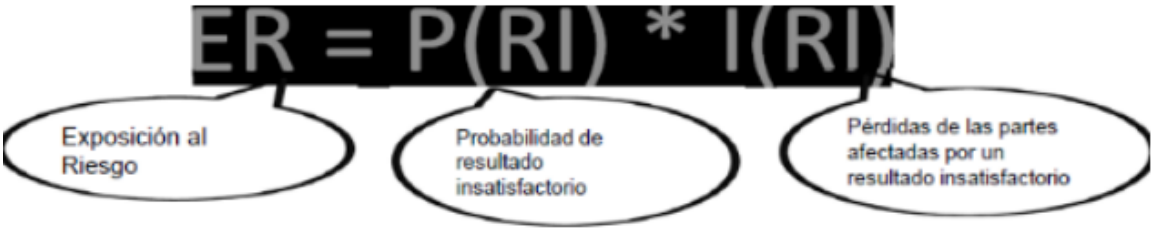
- **Objetivo del proyecto:**
  - Claro
  - Alcanzable
- **Definición del Alcance del Proyecto:** Todo el trabajo y SOLO el trabajo que se debe hacer para cumplir el objetivo.
- **Definición de Proceso y Ciclo de Vida**
- **Estimación:** En un proyecto de desarrollo de software, se trata de estimar para predecir el valor de un elemento relacionado con el proyecto o con el

producto, por ejemplo, el tiempo que va a llevar, qué costo va a tener el sistema, cuántas personas necesito para realizar el desarrollo, etc. En un enfoque tradicional, esta estimación de valores es realizada únicamente por el líder de proyecto. En la metodología tradicional, existe un orden definido y recomendado para realizar las estimaciones:

1. Tamaño del Producto: Para estimar el tamaño de un proyecto se utilizan los requerimientos tomados en la etapa de requisitos. Esta estimación es una de las más importantes a realizar, ya que el aumento o disminución del tamaño del producto, afecta de gran manera a otros aspectos, por ejemplo, el esfuerzo aumenta drásticamente a medida que las líneas de código de un proyecto crecen, lo mismo sucede con la estimación de costos y de tiempos (calendarización). Esto es debido a que un aumento del tamaño trae consigo un aumento de complejidad del software, que puede repercutir de muchas maneras en el desarrollo.
2. Esfuerzo: Refiere a la cantidad de horas persona lineales haciendo una actividad por vez, dentro del desarrollo del producto. Esta estimación intenta responder cuántas horas lineales serán necesarias para construir el producto, previamente estimado su tamaño.
3. Tiempo: La estimación del calendario del proyecto permite responder al cuándo se van a realizar las tareas definidas. En el momento de la estimación, no es necesario realizar una calendarización muy específica a nivel de día, sino que se escala a tareas a realizar en, por ejemplo, un mes del año o incluso un trimestre.
4. Costos
5. Recursos Criticos

- **Gestión de Riesgos:** El riesgo es la probabilidad de ocurrencia de algo que impacte negativamente. Tambien es el problema esperando para suceder.

  Se calcula como:



Riesgo = Probabilidad * Impacto

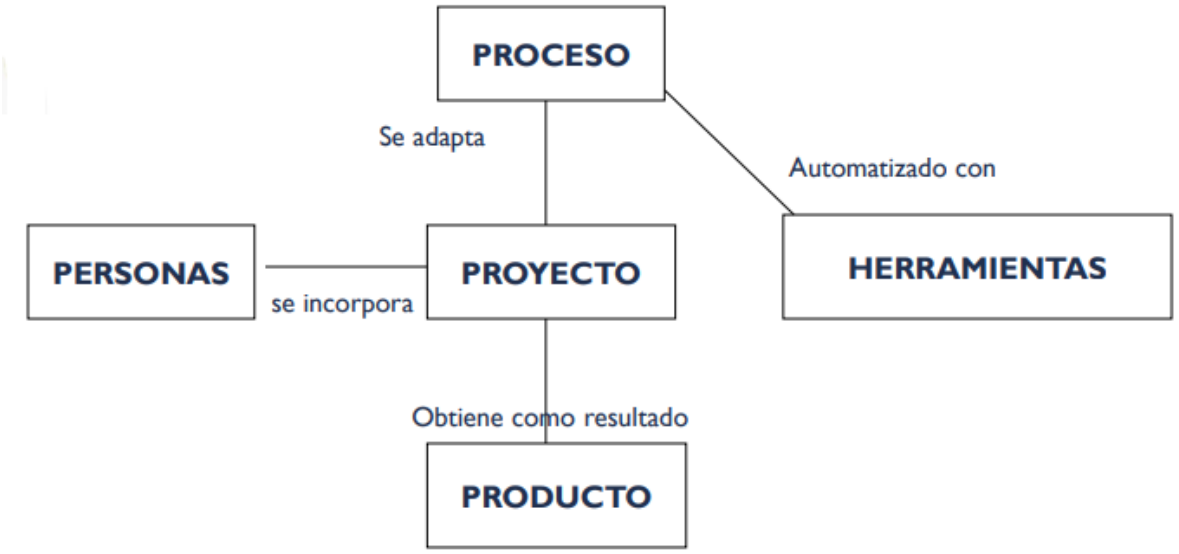
  - Plan de Contingencia: Que hacer si el riesgo se convierte en problema.
  - Plan de Mitigacion: Eliminar el riesgo.

- **Asignación de Responsabilidades | Recursos:** La asignación de responsabilidades a personas (no le agrada el concepto de recurso humano), permite asignar roles a los integrantes del equipo de trabajo. En el entorno de procesos definidos (PUD, por ejemplo), estos roles se ven definidos en el concepto de trabajadores.
  Dentro de un mismo proyecto, una persona puede desenvolverse en más de un rol, ya que puede que el presupuesto del desarrollo de software no sea lo suficientemente grande para permitirse tener una persona distinta por rol, o bien, el proyecto no sea lo suficientemente grande o complejo para justificar dicha adición.
  Para dejar explícitos los roles y quienes son las personas que los cumplen, generalmente se construye una tabla, la cual contiene la información de la persona, el rol y responsabilidades de esta en el contexto del proyecto.
  Se debe hacer mucho foco en la selección del personal de un proyecto, ya que las capacidades de los integrantes del equipo afectan a la calidad del software y al correcto desarrollo del proyecto en términos de tiempo, por ejemplo, ya que el desarrollo de software es una actividad HUMANO-INTENSIVA.
- **Programación de Proyectos | Calendarizacion:** En esta etapa se trata de definir en detalle, cada tarea que debe ser cumplida en el desarrollo. Se toma como base lo definido en la estimación del calendario realizada previamente, pero se refina al máximo detalle posible cada actividad. Cuando se habla de detallar las tareas, se hace referencia a descomponer el proceso de desarrollo en actividades refinadas a nivel de días, identificando quien la realiza, cuando debe realizarse y cuánto esfuerzo debe llevar en forma teórica.
- **Definición de Controles:** En la planificación de los controles del desarrollo, se toma como base las métricas ya definidas, y verifica que todo se está haciendo en concordancia con lo establecido. En esta planificación se definen reuniones periódicas, reportes o informes necesarios, etc
- **Definición de Métricas:** Las métricas se definen como una medida numérica que aporta visibilidad sobre el avance o estado del proyecto, proceso o producto en un momento determinado del desarrollo. Al realizar

la planificación del proyecto, es necesario definir de forma clara y no ambigua, cada una de las métricas asociadas a cada dominio (producto, proyecto o proceso) en conjunto con la forma de calcularlas.

- **Reuniones e informes.**

Vinculo proceso-proyecto-producto en la gestión de un proyecto de desarrollo de software



Se dice que el proceso, automatizado con herramientas, se adapta al proyecto, al cual
se incorporan personas, y de este se obtiene como resultado un producto.

En el enfoque tradicional, previo a la definición del proyecto, es necesario determinar los objetivos y el ámbito del producto para de esta manera poder realizar las estimaciones pertinentes. Los objetivos identifican las metas globales del producto sin especificar como se van a lograr. El ámbito del producto define las funciones y comportamientos que caracterizan al producto que utilizara cliente.

El proceso proporciona un marco conceptual en donde se establece un plan completo para el desarrollo del software. Define como se va a desarrollar el software, estableciendo un conjunto de actividades.
Las disciplinas de gestión permiten que las actividades del marco conceptual se adapten a las características del proyecto de software y a los requerimientos del equipo.
Las disciplinas de soporte son independientes del marco conceptual y

recubren al modelo del proceso, es decir que atraviesan todas las actividades del proceso de desarrollo.
Se debe definir el modelo de proceso a utilizar: secuencial, iterativo o recursivo. Luego, el marco conceptual que incluye las actividades fundamentales del desarrollo del software se adapta al modelo elegido.
Un proyecto de desarrollo está integrado por un equipo de trabajo, dentro del cual deben estar los roles bien definidos, para que cada uno de los integrantes asuma la responsabilidad que le corresponda, y se tengan en claro los alcances de cada uno de estos roles. El equipo va a basar su trabajo en el proceso adoptado, en el cual se van a tomar sólo aquellas actividades y formas de trabajo que deberán utilizar y seguir para lograr el desarrollo del producto/servicio deseado.
Las personas son la parte más importante, debido a que el desarrollo de software es una actividad humano-intensiva. Si se adopta el “mejor proceso” para el desarrollo de un software, y además se destina mucho esfuerzo a planificar el proyecto de desarrollo, de nada servirá si no tenemos un equipo capacitado que reúna las habilidades y herramientas necesarias para el desarrollo del mismo.
En el desarrollo de un proyecto, se busca utilizar diversas herramientas que se adapten y permitan automatizar o facilitar las actividades que están definidas en el proceso adoptado.

No Silver Bullet

- Contrastar progreso del hardware (algo tangible, construible, que se puede mejorar en su construcción), en comparación con el software (intangible, complejo inherentemente).
- Dificultades que plantea:
    - Esenciales: Todo aquello que trae aparejado el construir software: conceptos abstractos, estructuras de datos, algoritmos, invocación de funciones, etc. (carece de precisión y detalles). 4 aspectos la caracterizan:
        - Complejidad:
            - Propiedad inherente, debido a la abstracción del software
            - Al crecer el tamaño del software, su complejidad crece de manera exponencial, no lineal

        - Esto conduce a muchos problemas: dificultad de comunicación entre miembros de equipos, deficiencias en el producto, errores en cumplimiento de fechas, excesos en costos, etc.
    - Conformidad:
        - El software debe realizar aquello que el cliente necesita, esto es: debe adecuarse a sus requerimientos.
    - Mutabilidad:
        - El software está sujeto a constantes cambios. A diferencia de objetos tangibles, el software muta y sufre cambios a lo largo del tiempo. En cambio, en muchos objetos esto no es así, ya que se reemplazan por nuevos productos (no tomo el objeto y lo modifico, creo uno nuevo).
    - Invisibilidad:
        - Al ser algo no visualizable, se dificulta mucho trabajar con representaciones
          geométricas, como puede ser con estructuras mecánicas. Los grafos son una buena herramienta que permiten modelar muchas situaciones de un software.
  - Accidentales: Tiene que ver con aquellos aspectos que dificultan la construcción del software, pero han sido solucionados a lo largo del tiempo. Esas soluciones son:
    - Lenguajes de programación de alto nivel: permiten abstraer el proceso de construcción del software, independizándose del hardware en el que se ejecutarán. Esto elimina la complejidad propia de una máquina, que nada tiene que ver con el software que se desea construir.
    - Tiempo compartido: reducción de los tiempos de respuesta de un software. Esto permite no desviar el foco de lo que se desea construir.
    - Entornos de programación unificados: facilitan la construcción del software al integrar librerías, formato de archivos, frameworks, etc.
- Resolucion de Dificultades Esenciales:
  - Comprar, antes que construir: Explorar en el mercado soluciones que se puedan conseguir para reemplazar lo que se quiera construir. Si existe

un mercado que lo ha demandado, comprarlo siempre será más barato que construirlo. Esto se debe a que al tener una gran cantidad de clientes que lo compran, el costo de ese software se amortiza.
Mirarlo como que el uso de N copias de software, multiplica la productividad de ese software N veces.
Lógicamente este enfoque es difícil de utilizar para necesidades muy puntuales, donde no existe una porción de mercado que lo haya demandado previamente.

- Usar prototipos y refinar requerimientos: la parte fundamental de construir software es saber qué se debe construir. La extracción iterativa de requerimientos y su posterior refinamiento permiten tomar las necesidades de los clientes en sucesivas iteraciones, debido a que ellos mismos no saben expresar ni conocer muchas veces cuáles son esas necesidades que tienen. Además, es prácticamente imposible que sepan expresar con detalle y precisión todos sus requerimientos para el software que
necesitan.
El desarrollo de prototipos rápidos permiten cubrir esa extracción iterativa de requerimientos, al simular interfaces y funciones principales de un software que se necesita construir. Esto permite obtener un feedback valioso del cliente, para que todos comprendan mejor cuáles son las funcionalidades que se necesitan, y cómo se deben ejecutar.
- Desarrollo incremental: (crecer y no construir software): Incrementar el software añadiendo funcionalidades mientras estas se prueban, usan y testean.
Tener un software con pocas funcionalidades, pero funcionando, es más animador que tener un avance de un software complejo que no funciona.
- Personas capacitadas: que sigan buenas prácticas. Esto permite tener un buen equipo, pero, al fin y al cabo, desarrollar software requiere de creatividad, por lo que priorizar la creatividad en un equipo puede traducirse en grandes resultados (estructuras rápidas, pequeñas, simples construidas con
menos esfuerzo).
Esto plantea que tener en un equipo con buenos diseñadores, es tan importante como tener buenos líderes que dirijan ese equipo. Por lo que se debe recompensarlos de la misma forma, debido a su gran valor

para el desarrollo del
software.

- Resolución de Dificultades Accidentales:
  - Programación orientada a objetos: Permite a los diseñadores del software, abstraerse de tecnicismos para expresar su diseño simplemente en un lenguaje de máquina. Esto facilita el diseño, pero sólo elimina una dificultad accidental, no elimina la propia dificultad esencial de diseñar software.
  - Inteligencia Artificial: Si bien la IA permite resolver muchas cuestiones que reemplazan acciones humanas, siempre tiene que haber un equipo de personas qué programe y "enseñe" a esa IA cómo actuar. Con lo cual, el factor humano no se reemplaza del todo.
  - Sistemas expertos: son sistemas que aplican la IA y permiten sugerir diferentes situaciones como estrategias de Testing, optimizaciones de código, sugerencias en nombrado de variables, etc.
    Igualmente, no se quita el factor humano ya que se necesita encontrar expertos y desarrollar técnicas eficientes para extraer lo que ellos saben para destilarlo dentro de bases de reglas de estos sistemas.
    El prerrequisito esencial para construir un sistema experto es tener un experto.
  - Programación automática: esto se planteó como un reemplazo a los programadores, pero en lugar de ser eso, terminan siendo un lenguaje de más alto nivel que el programador debe parametrizar.
  - Programación gráfica:
    - Permite programar a un nivel más alto de abstracción que la programación tradicional, pero sigue estando el factor humano presente, con lo que solo se logra dar una mayor abstracción al programador sin que este tenga la necesidad de aprender sintaxis del lenguaje.
  - Verificación de programas:
    - Lo que se logra con la verificación de programa, es establecer que un programa pasa las pruebas matemáticas de verificación que se proveen.

        - No permite verificar que se cumplan todas las necesidades del cliente que fueron previstas en el software en forma de funcionalidades, sino que verifica que las funcionalidades desarrolladas no tengan errores.
        - Además, las pruebas matemáticas pueden también estar sometidas a errores.
    - Environments and tools:
        - La mayoría de los nuevos entornos inteligentes tan solo nos permiten librarnos de errores semánticos y sintácticos simples.
    - Estaciones de trabajo poderosas:
        - El incremento de potencia y memoria de las estaciones de trabajo individuales no soluciona el déficit en el desarrollo el software.
        - La edición de programas y documentos son soportadas totalmente por las máquinas de hoy en día.
        - Una mejora en tiempos de compilación no representa un avance significativo.

UNIDAD 2

PARCIAL 1

Gestión de Producto

- Un producto nuevo tiene una hipotesis de valor unico: **PRODUCTO / SERVICIO SERA UNICO.**

MVP

Producto Minimo Viable

Version de un nuevo producto creado con el menor esfuerzo posible.

Se crea para probar la hipotesis y se utiliza para ver que tan viables es un producto para un subconjunto de potenciales clientes.

Mediante su uso obtenemos aprendizaje validado por los clientes.

MVF

Caracteristica Minima Viable

Version mini del MVP

Característica a pequeña escala que se puede construir e implementar rápidamente, utilizando recursos mínimos, para una población objetivo para probar la utilidad y adopción de la característica.

Un MVF debe proporcionar un valor claro a los usuarios y ser fácil de usar.

MMF

Caracteristicas Minimas Comercializables

Es la pieza más pequeña de funcionalidad que puede ser liberada
- tiene valor tanto para la organización como para los usuarios.
- Es parte de un MMR or MMP.

El objetivo del MMF es aportar valor, supone una alta certeza de que existe valor en esta area y que sabemos cual debe ser el producto para proporcionar este valor.

MMP

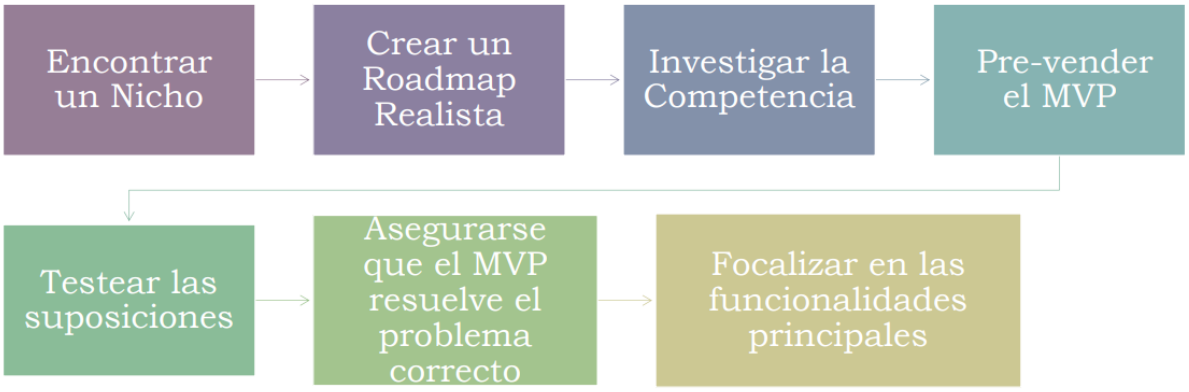
Producto Minimo Comercializable
Es la versión más pequeña y simple de un producto que se puede lanzar al mercado con suficiente valor como para atraer a los usuarios y ser comercializable. A diferencia del MVP, el MMP debe tener un **nivel suficiente de calidad y funcionalidad** para que los usuarios lo acepten y lo usen de forma sostenida.

MMR

Release Minima Comercializable

Es el conjunto mínimo de funcionalidades que debe tener una nueva versión de un producto para ser **lanzada y comercializada** en el mercado.
MMP = MMR1

Como preparar un MVP?



Que son las Hipotesis de Valor y de Crecimiento?

- Hipotesis del Valor: Prueba si el producto realmente esta entregando valor a los clientes despues de que comienzan a usarlo.
- Hipotesis de Crecimiento: Prueba como nuevos clientes descubriran el producto.

Requerimientos Ágiles

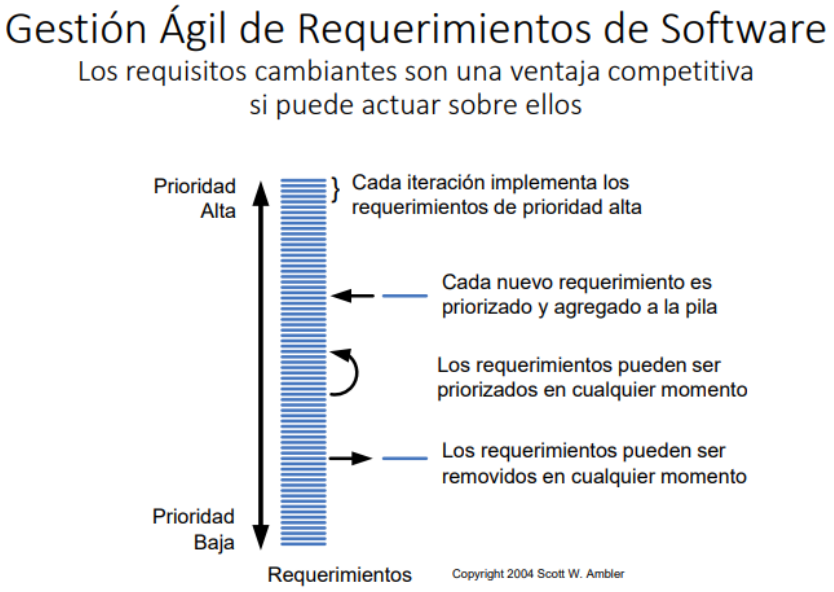
Que son los Req. Agiles?

Un requerimiento ágil es una especificación de una necesidad o funcionalidad que se define de manera flexible y colaborativa dentro de un enfoque ágil de desarrollo de software. En lugar de ser un documento extenso y detallado, los requerimientos ágiles suelen ser más breves y se enfocan en la descripción de la funcionalidad desde la perspectiva del usuario.

3 Pilares de los Requerimientos Agiles:

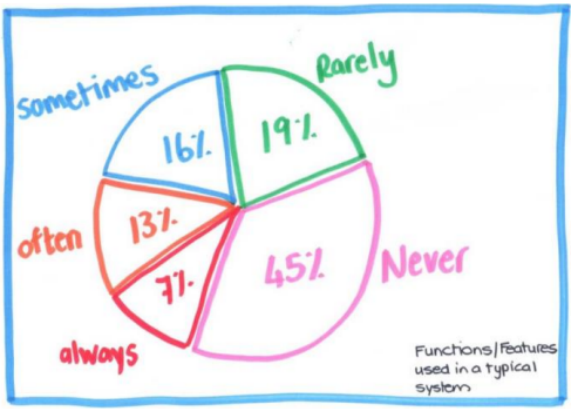
- Usar el valor de negocio para construir el producto correcto: Lo único importante es dejar contento al cliente con software funcionando que le sirva, por lo que, el foco de este enfoque de gestión ágil apunta a construir valor de negocio.
- Determinar que es “sólo lo suficiente”: hace referencia a que los requerimientos deben ser encontrados de a poco, no buscar todos los requerimientos desde un inicio y quedarnos solo con eso. Este pilar postula que se tiene que empezar con lo mínimo necesario para generar realimentación y alimentar el empirismo y a partir
de allí, se avanza continuamente.

- Usar historias y modelos para mostrar que construir: la parte de desarrollo de requerimientos está muy enfocada a la idea de construir junto con el cliente, respetando el principio ágil de “técnicos y no técnicos trabajando juntos”. Se busca trabajar con un referente que esté del lado del negocio, que tenga el conocimiento del producto y que trabaje junto con el equipo.



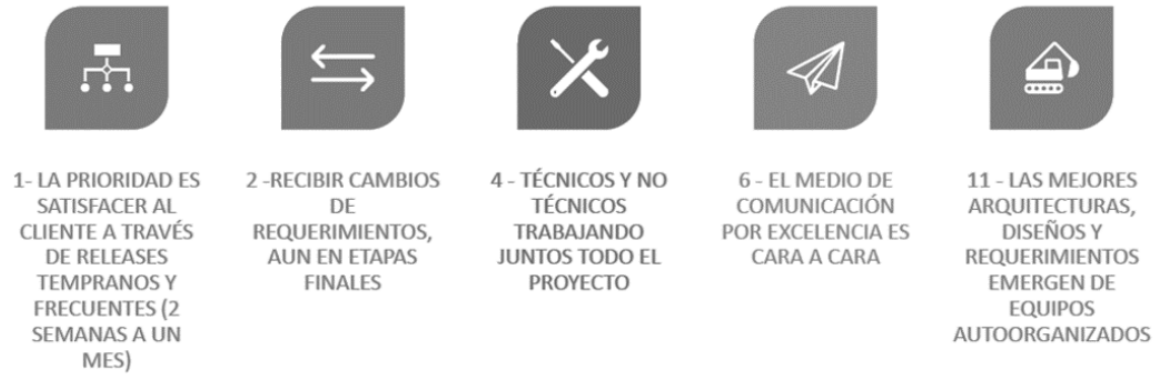
BRUF - Big Requirements Up Front

La imagen adjuntada expresa la frecuencia con la que los usuarios utilizan determinadas funcionalidades de un software. A lo que se apunta con este concepto de BRUF es a notar que, en una primera instancia de un proyecto de software, desarrollar sólo unas pocas funcionalidades de gran valor e importancia para el cliente, puede cubrir gran parte de sus necesidades, a medida que se desarrolla el resto de los requerimientos.



Una caracteristica de los requerimientos agiles es la de que usan un enfoque JUST IN TIME, donde analizan los requerimientos solo cunado se necesiten.

Principios Agiles relacionados con Requerimientos Agiles



Como representamos los Reqs Agiles?
⬇

User Stories

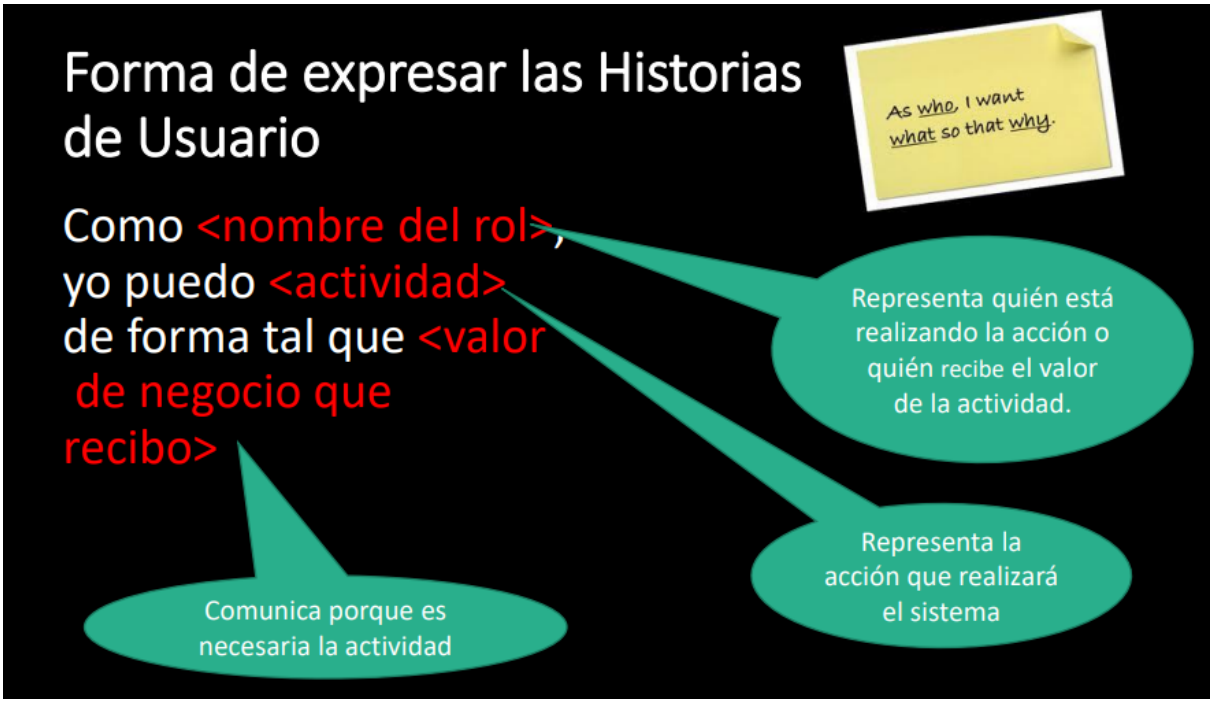
Que es una User Story?

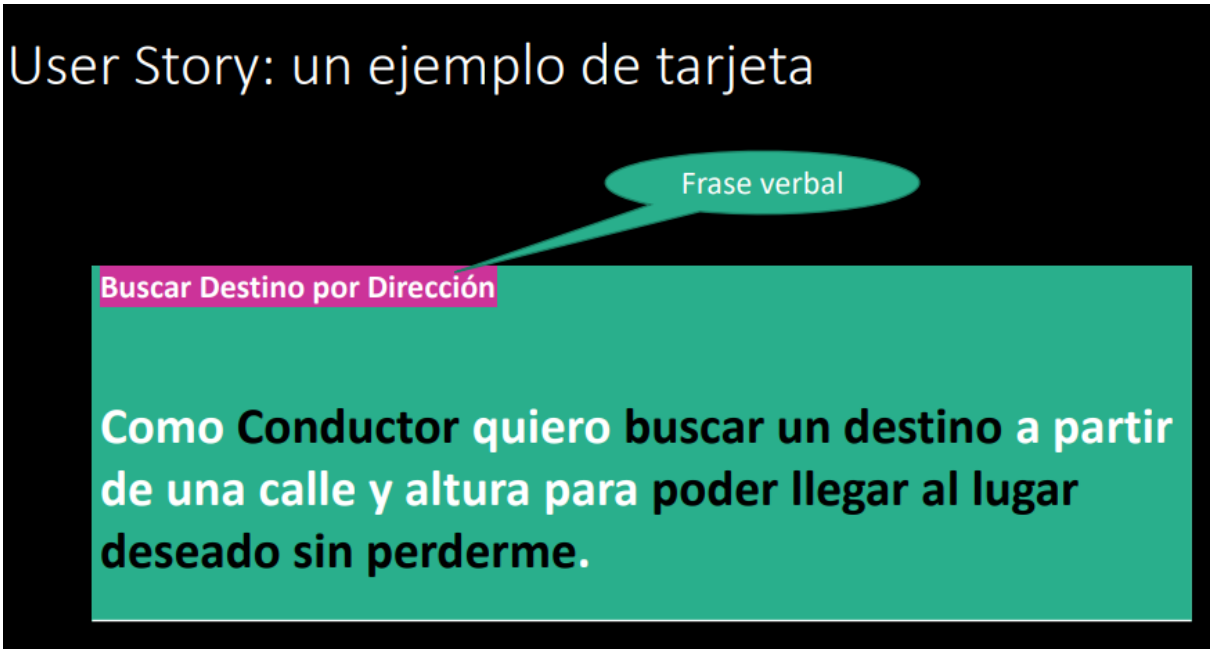
Es una herramienta utilizada en las metodologias agiles para capturar una necesidad o requerimiento del usuario de manera concisa y centrada en el valor que proporciona. Se redacta desde la perspectiva del usuario final.

Las user stories estan a nivel de los requerimientos de **NEGOCIO.**

⚠

- No son especificaciones detalladas de requerimientos (como los casos de uso)
- Son expresiones de intención, “es necesario que haga algo como esto…”
- No están detallados al principio del proyecto, elaborados evitando especificaciones anticipadas, demoras en el desarrollo, inventario de requerimientos y una definición limitada de la solución.
- Necesita poco o nulo mantenimiento y puede descartarse después de la implementación.
- Junto con el código, sirven de entrada a la documentación que se desarrolla incrementalmente después.





💡 Si una US se hace muy grande se convierte en una EPICA y si es mas grande todavia se convierte en un TEMA, el cual es un agrupador de US.

LAS 3Cs de USER STORY

- **CONVERSACION:** Este es el diálogo entre el equipo de desarrollo y los interesados.
- **CARD:** Representa la historia de usuario en un formato físico o digital.
- **CONFIRMACION:** Se refiere a los criterios de aceptación o condiciones que deben cumplirse para que la historia se considere completa.

💡 Las US son porciones verticales ya que abarcan todo el diseño y arquitectura.

Que son los Criterios de Aceptacion?

Son una parte de la US y definen los limites para la misma, ayudan a que los PO respondan lo que necesitan para que la US provea valor.

Son criterios que nos ayudan a evitar el rechazo por parte del PO.

Pueden abarcar Requerimientos No Funcionales si se relacionan con la US.

Caracteristicas de buenos criterios:

- Definen una intencion no una solucion.
- Son independientes de la implementacion.
- Relativamente de alto nivel, no es necesario que se escriba cada detalle.

Que son las Pruebas de Aceptacion?

Expresan detalles resultantes de la **CONVERSACION.**

Complementan la US.

Niveles de Abstraccion de una US

1. User Story: Lista para ingresar a una iteracion.
2. Epica: US muy grande, el tamaño es relativo a si ingresa a una iteracion o no. Si la US no entra en la iteracion es una Epica.
   Una epica debe ser divida en US mas pequeñas.
3. Temas: Conjunto de US agrupadas, debido a que conceptualmente esta relacionadas. Se las agrupa para tratarlas com una entidad simple a la hora de estimar o planificar.

Que es una Spike?

Tipo especial de historia, utilizado para quitar riesgo e incertidumbre de una US u otra faceta del proyecto.

Se clasifican en Tecnicas y Funcionales.

Tecnicas:

Utilizadas para investigar enfoques tecnicos en el dominio de la solucion.

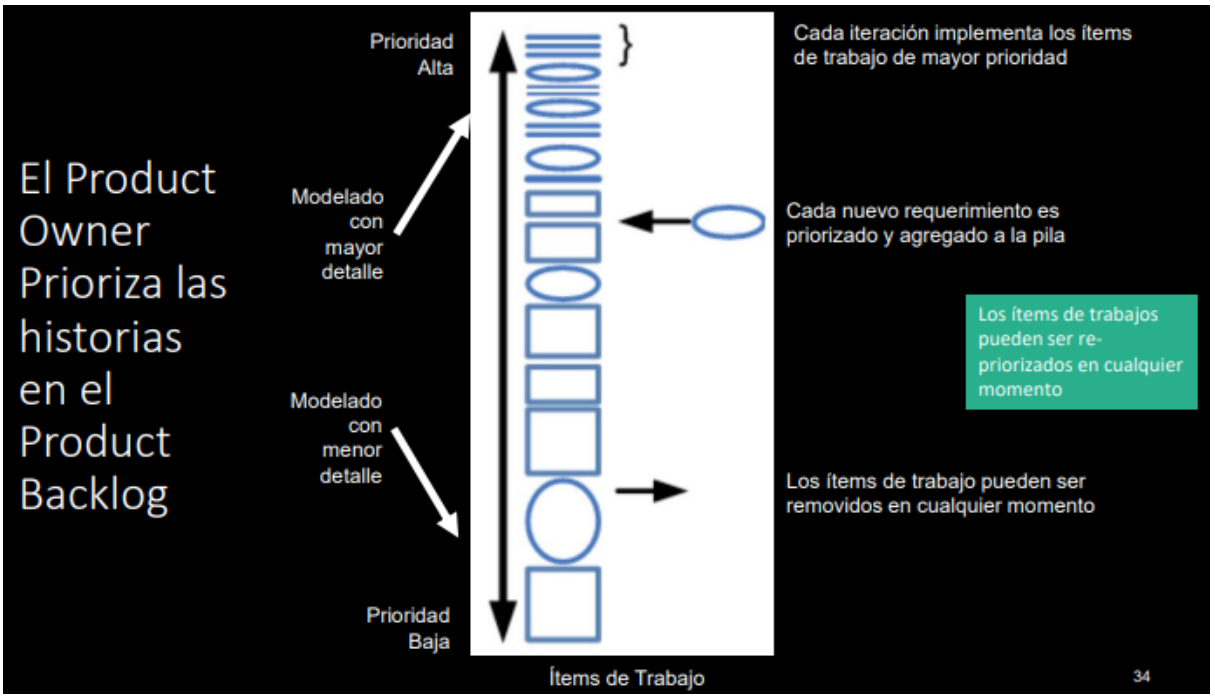
Cualquier situacion en la que el equipo necesite una comprension mas fiables ante de comprometerse a una nueva funcionalidad en un tiempo fijo.

Funcionales:

Utilizadas cuando hay una cierta incertidumbre respecto de como el usuario interactuara con el sistema.

Que es el Product BackLog?

Es la lista priorizada completa de funcionalidades, mejoras, correcciones de errores y otros requisitos que se necesitan para el producto. Es administrado por el Product Owner y se actualiza continuamente a medida que se reciben nuevos requerimientos y se obtiene feedback del cliente. Cada ítem en el product backlog suele estar representado como una user story o un ítem de trabajo.



Estimaciones Ágiles

Nos hablan de una incertidumbre o falta de informacion.

Involucran PROBABILIDAD → NO son certezas o compromisos

Que es una Estimacion Agil?

Es el proceso de evaluar el esfuerzo, complejidad e incertidumbre que tenemos a la hora de ejecutar una tarea.

> 💡 Se estima en COLABORACION con el equipo

Estimacion Relativa:

Son mejores que las estimaciones absolutas debido a que las personas no son buenas estimando en terminos absolutos, en cambio somos buenos

comparando cosas y estas comparaciones son mas rapidas.

> 💡 Las US se estiman usando STORY POINTS

Tamaño

Es la medida de la cantidad de trabajo necesaria para producir una story.

El tamaño indica:

- Cuán compleja es una feature/story.
- Cuánto trabajo es requerido para hacer o completar una feature/story.
- Y cuán grande es una feature/story

> ⚠️ Tamaño NO es Esfuerzo

Normalmente en las US se utiliza como escala del tamaño la serie de fibonacci.

Que es un Story Point?

Es una unidad de medida especifica del equipo, de la complejidad, riesgo y esfuerzo.

La complejidad de una feature/ story tiende a incrementarse exponencialmente.

Si cuesta estimar una US pasa a ser una Spike.

Que es la Velocidad?

Es una medida del progreso de un equipo, se calcula sumando el numero de story points de cada US que el equipo completa durante la iteracion.

Solo valen US que estan completas.

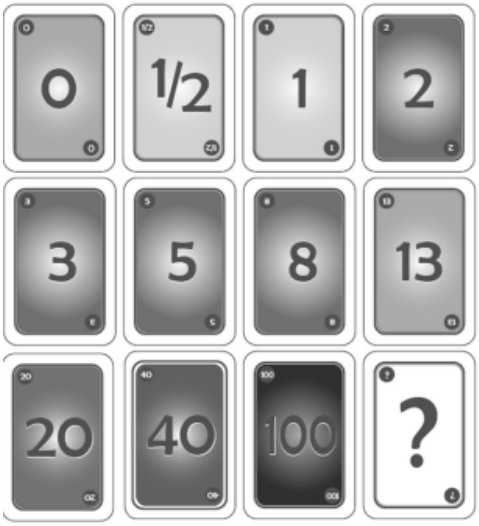
Que es el Poker Estimation?

Es una tecnica para estimar que combina la opinion de experto, analogia y desegregacion.

Los participantes del planning poker son desarrolladores:

> “Las personas más competentes en resolver una tarea deben ser quienes las estiman”

¿Cómo “decodificar” las estimaciones?



- **0**: Quizás ud. no tenga idea de su producto o funcionalidad en este punto.
- **1/2, 1**: funcionalidad pequeña (usualmente cosmética).
- **2-3**: funcionalidad pequeña a mediana. Es lo que queremos. ☺
- **5**: Funcionalidad media. Es lo que queremos ☺
- **8**: Funcionalidad grande, de todas formas lo podemos hacer, pero hay que preguntarse sino se puede partir o dividir en algo más pequeño. No es lo mejor, pero todavía ☺
- **13**: Alguien puede explicar por que no lo podemos dividir?
- **20**: Cuál es la razón de negocio que justifica semejante story y más fuerte aún, por qué no se puede dividir?.
- **40**: no hay forma de hacer esto en un sprint.
- **100**: confirmación de que está algo muy mal. Mejor ni arrancar.

Requisitos para Poker Planning:

- Lista de stories a ser estimadas.
- Cada estimador tiene un mazo de cartas.

Pasos:

1. Determinar la US canonica que sera usada como comparacion con las otras US.
2. Se lee la story a estimar y se discute con todo el equipo.
3. Cada estimador selecciona una carta y pone la carta boca abajo, cuando todos pusieron sus cartas se dan vuelta.
4. Si todos los estimadores seleccionan el mismo valor entonces ese es el estimado, si no se discute comenzando por quienes dieron el valor mas alto y mas bajo. Despues de la charla se vuelve al paso 3.
5. Se toma la prox. US.

ESTIMACION BASADA EN EXPERIENCIA

- Datos historicos.

- Juicio Experto:
  - Puro: Un experto estudia las especificaciones y hace su estimacion, se basa principalmente en los conocimientos del experto y si este desaparece la empresa deja de estimar.
  - Wideband Delphi: Un grupo de personas informadas estiman, se producen discuciones y se llega a un acuerdo.
- Analogia.

PARCIAL 2

UNIDAD 3

PARCIAL 1

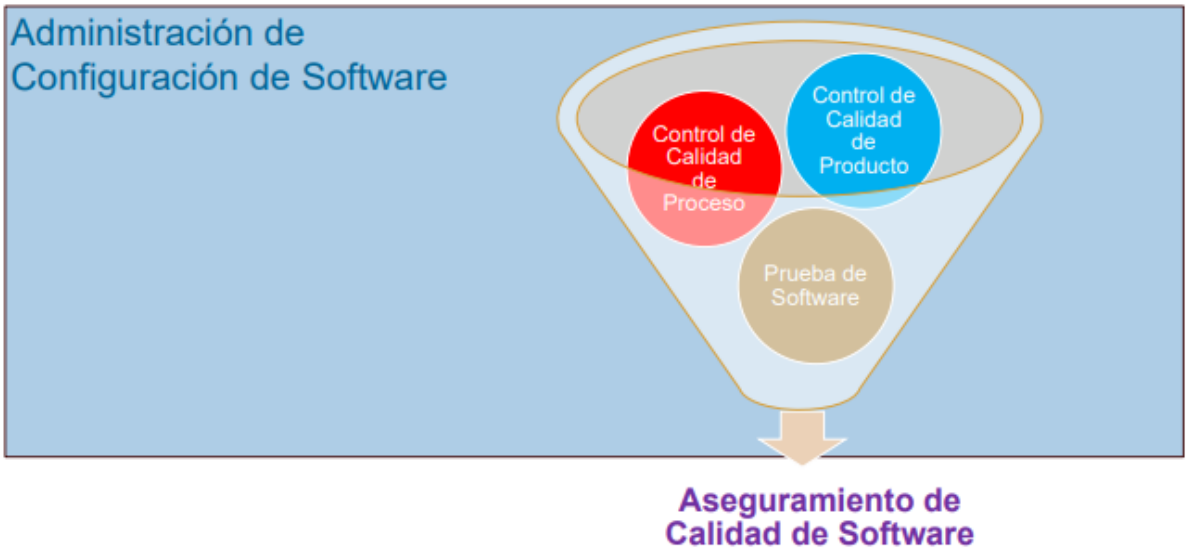
SCM - Software Configuration Management

Que es el SCM?

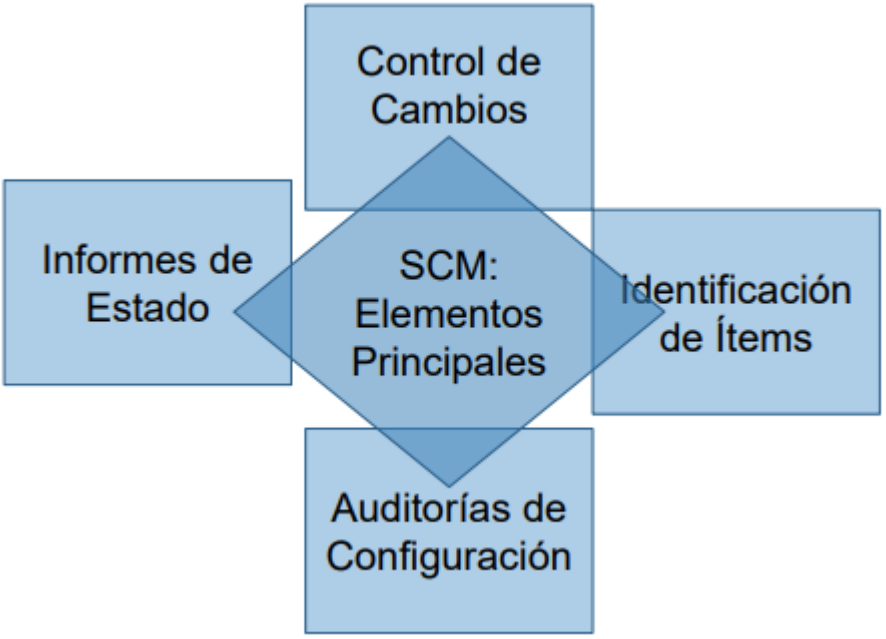
Es una disciplina de soporte dentro de las disciplinas de desarrollo de software que actua como protectora (paraguas) del software, que vela por mantener la integridad de los items de configuracion, osea la integridad del producto.

Aplica dirección y monitoreo administrativo y técnico a: identificar y documentar las características funcionales y técnicas de los ítems de configuración, controlar los cambios de esas características, registrar y reportar los cambios y su estado de implementación y verificar correspondencia con los requerimientos.

> 💡 Es transversal a todo el proyecto.



Actividades Fundamentales



1. Identificacion de Items:
    a. Identificacion univoca de cada Item de config.
        i. Items de Producto
        ii. Items de Proyecto
        iii. Items de Iteracion.

b. Convenciones y reglas de nombrado.

c. Definicion de la estructura del repositorio.

d. Ubicacion dentro de la estructura del repositorio.

2. Control de Cambios:

   a. Tiene su origen en la solicitud de cambios en uno o varios items de configuracion que se encuentran en la linea base.

   b. Procedimiento formal que involucra diferentes actores y una evaluacion del impacto del cambio.

   c. Comite de Cambios:

      Está formado por representantes de todas las áreas involucradas en el desarrollo:

      - Análisis
      - Diseño
      - Implementación
      - Testing
      - Otros interesados

3. Auditorias de Configuracion de Software:

   a. Auditoría física de configuración (PCA): Asegura que lo que está indicado para cada ICS en la línea base o en la actualización se ha alcanzado realmente.

   b. Auditoría funcional de configuración (FCA): Evaluación independiente de los productos de software, controlando que la funcionalidad y performance reales de cada ítem de configuración sean consistentes con la especificación de requerimientos.

Sirven a dos procesos basicos:

- Validacion: El producto responde correctamente al problema.
- Verifiacion: El producto cumple con los objetivos preestablecidos definidos en la documentacion de las lineas bases.

4. Informes de Estado:
    a. Registros de la evolucion del sistema.
    b. Incluyen reportes de rastreabilidad de todos los cambios realizados.

Que es un Item de Configuracion?

Son todos y cada uno de los artefactos que forman parte del producto o del proyecto, que pueden sufrir cambios o necesitan ser compartidos entre los miembros del equipo y sobre los cuales necesitamos conocer su estado y evolucion.

Que es una Version?

Se define desde el punto de vista de la evolucion, como la forma particular de un artefacto en un instante o contexto dado.

El control de versiones se refiere a la evolucion de un unico item de configuracion o de cada item por separado.

Pueden representarse graficamente como Grafos.

Que es una Variante?

Es una version de un item de configuracion que evoluciona por separado.

Representan configuraciones alternativas.

Que es la Configuracion de Software?

Un conjunto de items de configuracion con su correspondiente version en un momento determinado.

Que es un Repositorio?

Contenedor de items de configuracion que mantiene la historia de cada item con sus atributos y relaciones.

- Repo Centralizado: Un servidor que contiene todos los archivos.
- Repo Descentralizado: Cada cliente tiene una copia exactamente igual del repositorio completo.

Que es una Linea Base?

Conjunto de items o una version que se encuentra estable. Se marcan lineas bases mediante tags (etiquetas).

No es lo mismo que la version del producto.

Permiten ir atrás en el tiempo y reproducir el entorno de desarrollo en un momento dado del proyecto.

> 💡 El Comite de Cambios son quienes se encargan de los cambios en la linea base.

Existen dos tipos de linea base:

- Operacionales: Contiene una versión de producto cuyo código es ejecutable, han pasado por un control de calidad definido previamente. La primera línea base operacional corresponde con la primera Release. Es la línea base de productos que han pasado por un control de calidad definido previamente
- De Especificacion o Documentacion: Son las primeras línea base, dado que no cuentan con código. Podría contener el documento de especificación de requerimiento. Son de requerimientos, diseño.

Que es una Rama?

Es una bifurcacion en el desarrollo, sirven para experimentar sin afectar al producto principal.

Manifiesto Agil

Principios del Manifiesto Ágil

Seguimos estos principios:

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

2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladorestrabajamos juntos de forma cotidiana durante todoel proyecto.
5. Los proyectos se desarrollan en torno a individuosmotivados. Hay que darles el entorno y el apoyo quenecesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicarinformación al equipo de desarrollo y entre susmiembros es la conversación cara a cara.
7. El software funcionando es la medida principal deprogreso.
8. Los procesos Ágiles promueven el desarrollosostenible. Los promotores, desarrolladores y usuariosdebemos ser capaces de mantener un ritmo constantede forma indefinida.
9. La atención continua a la excelencia técnica y albuen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

PARCIAL 2

UNIDAD 4

PARCIAL 2

Anotaciones Extra

Notas Extra