Unidad 1: Ingeniería de software en contexto

Introducción a la ingeniería de software. ¿Qué es?

Software

El software es un conjunto de programas en conjunto con toda la documentación requerida para definir, desarrollar y mantener los programas ejecutables que se entregan al cliente (archivos de configuración utilizados para la ejecución, documentos para el usuario, documentos que describen la estructura del sistema), como así también las herramientas utilizadas para la construcción de este.

Otra forma de definir el software es como, la información o conocimiento que se presenta en distintos niveles de abstracción, desde los requerimientos del sistema (abstracto), hasta el código que ejecuta el mismo (detallado).

Ingeniería de Software

Es una disciplina que se encarga de todos los aspectos de la producción del software, desde la primera etapa de especificación del sistema hasta el mantenimiento del sistema después de que se pone en operación.

Esta ingeniería es importante ya que:

- Con mayor frecuencia se requiere producir de manera rápida y económica sistemas confiables.
- Resulta más barato a largo plazo utilizar métodos y técnicas de ingeniería de software para los sistemas de software. La mayoría de los costos consisten en cambiar el software después de que se pone en operación.

Además, la ingeniería de software se apoya en varias realidades, entre ellas:

- En primer lugar, se debe hacer un esfuerzo para entender el problema antes de desarrollar una aplicación de software. Es decir, se debe entender el problema y las necesidades del cliente antes de proponer una solución.
- El diseño es una actividad crucial en el desarrollo de un software.
- Realizar un correcto diseño del software, trae consigo dos claros beneficios, calidad de software y facilidad de mantenimiento.

La ingeniería de software nace tras las crisis del software, en la cual existía (y existe) un conjunto de dificultades o errores ocurridos en la planificación, estimación de los costos, productividad y calidad de un software, debido, principalmente, a la baja eficacia que presentan una gran cantidad de empresas al momento de desarrollarlo.

Estado actual y antecedentes. La crisis del software.

El término crisis del software hace referencia a un conjunto de hechos relativos al software, planteados en la conferencia de OTAN en 1968 por Friedrich Bauer, quien recalco la dificultad para generar software libre de defectos, fácilmente comprensibles y que sean verificables. Sin embargo, este término fue utilizado anteriormente por Dijkstra en El Humilde Programador.

Causas

- La evolución de la tecnología del hardware mediante los circuitos integrados permitió la posibilidad del desarrollo de sistemas más grandes y complejos. El salto en el hardware no fue acompañado por un salto en el desarrollo del software, esto en gran parte se debe a que el software es una actividad humana cuyo producto es abstracto e intangible.

- En combinación con lo anterior, la demanda creciente de estos sistemas complejos, la subestimación de la complejidad que supone el desarrollo de software, los cambios a los que tienen que ser sometidos los productos para ser adaptados a las necesidades del cliente y la falta de una disciplina que intervenga en todos los aspectos de la producción del software fueron las causas de esta crisis.

En la actualidad, muchos de los sistemas desarrollados fracasan debido a fallas en el software, que principalmente son consecuencia de dos factores:

- Demandas crecientes: a medida que la tecnología y las técnicas de desarrollo evolucionan, la demanda de sistemas más grandes y complejos aumenta. Los usuarios necesitan que se construyan y distribuyan de manera rápida. La problemática aquí es que los métodos existentes (tradicionales) de ingeniería no permiten lograr esta "espontaneidad", por lo que se deben desarrollar y poner a prueba nuevas técnicas y métodos de gestión y desarrollo.

- Bajas expectativas: muchas de las compañías actuales de desarrollo de software no utilizan métodos de ingeniería de software en su trabajo diario. Por lo tanto, su software con frecuencia es más costoso y menos confiable de lo que debiera. La consecuencia, recae en la conformidad de las empresas y de los usuarios ante un software, es decir, las empresas se conforman con que el software solo "funcione", sin tener en cuenta el valor agregado que el software le debe aportar al negocio del usuario. Es necesaria una mejor educación y capacitación en ingeniería de software para solucionar este problema.

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)

Ejemplos de grandes proyectos de software fallidos y exitosos.

Proyectos fallidos

Proyectos exitosos

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

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.

Los modelos de proceso pueden incluir las mismas actividades, pero cada uno, desde una perspectiva particular, pondrá más énfasis en algunas respecto a otras y definirán flujos de procesos diferentes.

Además, establece los criterios que se utilizan para determinar cuándo se procede de una tarea a otra.

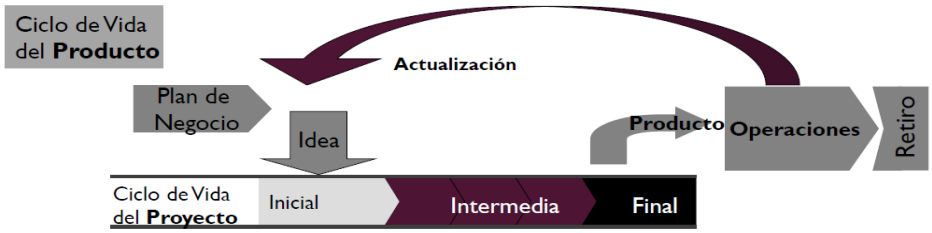
Los modelos de ciclos de vida especifican:

- Las fases del proceso (requerimientos, especificaciones, diseño, etc.)
- Orden en el cual se llevan a cabo.

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 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.



Clasificación de los ciclos de vida

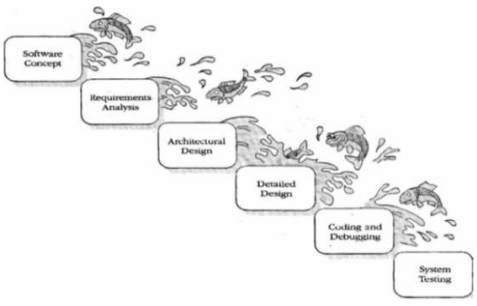
Modelo Secuencial

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.

Escenarios de proyectos donde elegirlo:



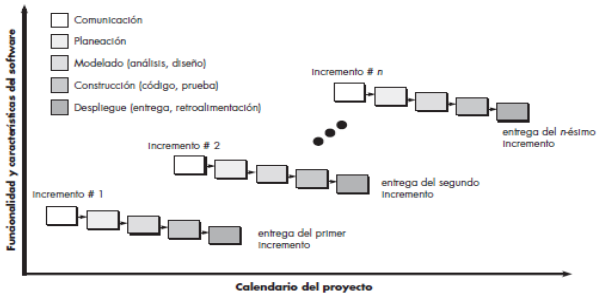
- El modelo en cascada puro ayuda a minimizar los gastos generales de planificación porque se puede hacer toda la planificación por adelantado. No proporciona resultados tangibles en el desarrollo de software hasta el final del ciclo de vida, pero la documentación que se genera a lo largo de este proporciona indicaciones significativas de progreso a lo largo del ciclo de vida.
- Es más sencillo utilizar un método de gestión común a lo largo de todo el proyecto, por lo que aún es común el uso de este modelo.
- El modelo en cascada puro funciona bien para productos en los que se tiene una definición clara y estable, y cuando se trabaja con metodologías técnicas claras.

Dificultades

- Los proyectos raramente siguen un flujo secuencial.
- El modelo secuencial exige que los requerimientos sean completamente explícitos desde un principio y esto en la realidad no suele suceder. No puede lidiar con la incertidumbre.
- El cliente obtiene una versión funcional del producto en etapas muy avanzadas del proyecto.
- Encontrar un defecto implica un gran rediseño en la solución, ya que está prácticamente todo hecho.
- Dependencia entre los equipos de trabajo. Un equipo no puede comenzar hasta que el otro no haya finalizado.
- El desarrollo está dirigido por un plan, y cada etapa general documentación para realizar un monitoreo constante contra el plan, por lo que esa documentación puede llegar a ser muy burocrática y excesiva.

Modelo 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.

Ventajas frente al modelo secuencial

- Se reduce el costo de adaptar los requerimientos cambiantes del cliente y el retrabajo es menor ya que en cada incremento se obtiene una retroalimentación que permite adaptar el modelo de las necesidades del cliente.
- El cliente es parte del proceso de desarrollo. En el modelo secuencial, el cliente debía esperar hasta la última instancia para recibir el producto y no era capaz de juzgar el avance del producto a través de documentos de diseño de software.
- Es posible que la entrega se a más rápida la entrega e implantación de software útil al cliente, aún si no se ha incluido toda la funcionalidad. Los clientes tienen la posibilidad de usar y ganar valor del software más temprano de lo que sería posible con un proceso en cascada.
- Muy útil para sistemas de requerimientos cambiantes.
- La especificación, desarrollo y validación están entrelazadas en lugar de separadas y aisladas.

Dificultades desde una perspectiva administrativa

- Se invisibiliza el proceso.
- Alto costo de documentar cada incremento.
- La estructura del software tiende a degradarse frente a una cantidad considerable de incrementos.

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. El modelo en espiral es un modelo recursivo, que consta de 4 etapas recursivas:

1. Determinar objetivos.
2. Identificar y resolver los riesgos.

3. Desarrollar y probar.

4. Planificar la próxima iteración.

Para llevarlo adelante, se subdivide el proyecto en varios mini proyectos, intentando resolver en cada uno los riesgos más relevantes hasta que no quede ninguno.

Dicho en otras palabras, se inicia con algo en forma completa, como una subrutina que se llama a sí misma e inicia nuevamente. Se presenta un prototipo que va mejorando con cada vuelta. Lo positivo, es que se generan productos independientes de la implementación, que pueden ser reusables en sistemas de características similares.

Dificultades desde una perspectiva administrativa

- Puede ser más costoso en tiempo y dinero readaptarlos para reutilizarlos para diferentes proyectos.
- La tecnología puede ser obsoleta.
- Puede carecer de mantenimiento o documentación, lo cual dificulta mucho las tareas.
- Se genera una versión del producto recién al final.

Administración de proyectos dependiendo en ciclo de vida

La elección de un ciclo de vida afecta de forma directa a la administración que se debe realizar sobre el proyecto, ya que cada uno de los modelos de proceso poseen características y enfoques distintos. Por ejemplo, si se realiza un análisis de riesgo y se determina que los requerimientos pueden variar mucho a lo largo del desarrollo, se debería elegir un ciclo de vida en el cual el cambio sea permisible, como los iterativos. En este caso, la administración y la planificación del proyecto se ve afectada en la elección del ciclo de vida.

Además, durante el desarrollo del proyecto, la gestión de este se realiza en consecuencia de esta elección, ya que no es lo mismo gestionar un proyecto que se desarrolla de manera iterativa, que uno que se desarrolla de manera secuencial o en cascada.

Procesos de Desarrollo Empíricos vs. Definidos.

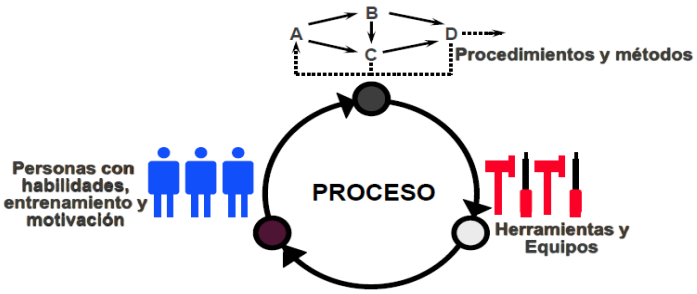
Proceso de desarrollo de software

Un proceso es una secuencia de pasos ejecutados para un propósito dado (IEEE), mientras que un proceso de software se define como un conjunto de actividades, métodos, prácticas y transformaciones que la gente usa para desarrollar o mantener software y sus productos asociados.

Las actividades del proceso de desarrollo son la especificación de requerimientos, el análisis, el diseño, la implementación y la prueba. El orden y la relación entre las diferentes actividades está determinada por la elección del ciclo de vida.

Se tienen tres factores determinantes del proceso:

- Procedimientos y Métodos: La adaptación de los procesos establecidos para su implementación, dependiendo del contexto, es esencial para la calidad resultante. Definir las actividades que se harán y cuáles no. Además, es relevante que esta información esté documentada para lograr transparencia. Es decir, tener definidos y escritos los procedimientos.



- Personas motivadas, capacitadas y con habilidades: Es factor primordial para obtener la calidad del producto del proceso, ya que éste se determina del esfuerzo de las personas. Sin ellas, no se puede lograr construir ningún

producto. Por ello, deben estar capacitadas y con habilidades para realizar sus tareas asignadas de la manera correcta y motivados, para lograr aún mayor eficiencia. Recordar que el software es una actividad humano-intensiva.

- Herramientas y Equipos: Materiales necesarios para llevar a cabo el proceso que nos permiten que las entradas se transformen en salidas. Se recomienda automatizar la mayor cantidad de actividades posibles, lo que mejora la eficiencia de las personas y puedan concentrarse en tareas que requieran de su capacidad.

Los procesos de software son complejos, por lo que no hay un proceso ideal; además, la mayoría de las organizaciones han diseñado sus propios procesos de desarrollo de software. Los procesos han evolucionado para beneficiarse de las capacidades de la gente en una organización y de las características específicas de los sistemas que se están desarrollando. Para algunos sistemas, como los sistemas críticos, se requiere de un proceso de desarrollo muy estructurado (proceso definido). Para los sistemas empresariales, con requerimientos rápidamente cambiantes, es probable que sea más efectivo un proceso menos formal y flexible (filosofía ágil).

Consideraciones

1. Estas actividades varían dependiendo de la organización y el tipo de sistema que debe desarrollarse, pero deben incluir: productos, roles, responsabilidades y condiciones

2. El proceso debe ser explícitamente modelado si va a ser administrado.

3. Un proceso de desarrollo se puede aplicar a varios ciclos de vida. La elección del ciclo de vida depende de la estrategia y este debe permitir seleccionar los artefactos a producir, definir actividades y roles, y modelar conceptos.

4. Las personas utilizan herramientas y equipos para llevar a cabo los procedimientos que componen el proceso de desarrollo.

Actividades fundamentales

1. **Especificación de software:** clientes junto con ingenieros definen el software a producir y sus restricciones.

2. **Desarrollo:** diseño y programación del software.

3. **Validación:** verificación para asegurar que es lo que el cliente quiere.

4. **Evolución:** donde se modifica el software para reflejar los requerimientos cambiantes del cliente y mercado.

Procesos Definidos

Los procesos definidos son considerados deterministas: ante la misma entrada se pretende que se obtendrá la misma salida. Asumen que, si se aplican una y otra vez, se obtendrán siempre los mismos resultados, a pesar de cambiar de equipo.

1. Están inspirados en las líneas de producción.
2. La administración y el control provienen de la predictibilidad del proceso. Es decir, el objetivo es lograr una repetibilidad, para poder obtener una previsibilidad de esfuerzo, riesgos, costos, etc. Asociados al desarrollo de software.
3. Puedo usar cualquiera de los ciclos de vida.

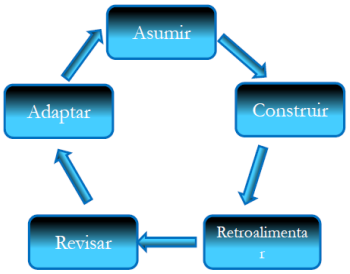
En la realidad, no es posible obtener el mismo resultado ante mismas entradas ni con el mismo equipo, ya que depende sumamente del contexto, momento y las personas, por más parecido sean estos factores.

La administración y control del proceso, en muchos de los casos, toman más importancia que el avance del software en sí (más importante la documentación del avance, que el avance en sí).

Procesos Empíricos

Los procesos empíricos se conforman en base a la experiencia interna y externa de las personas que intervienen en un contexto particular. NO es considerado determinista ya que no se asume que ante la aplicación del mismo proceso se obtendrán los mismos resultados. Dado que el factor diferencial es la experiencia sensible, en estos procesos se pueden obtener diferentes resultados dependiendo del contexto en el cual se apliquen.

1. Se basan en ciclos cortos de inspección y adaptación (retroalimentación), porque ante un análisis, se ajustan de mejor forma en aquellos dominios complejos donde prima la creatividad y/o complejidad.
2. La administración y el control es por medio de inspecciones frecuentes y adaptaciones para lograr buenas prácticas.
3. No se pueden combinar con cualquiera de los tipos de ciclos de vida vistos, se suele recomendar que sea el modelo iterativo-incremental, y el que se suele prohibir es el modelo secuencial debido a que no es viable después de 2 años (lo dice una teoría).



La experiencia que se obtiene en cada uno de los procesos no es extrapolable, es decir, que solo funciona en ese mismo proyecto, y se necesita adaptar los procesos a cada contexto en especial. Los conocimientos obtenidos en el proceso se deben hacer explícitos al grupo de desarrollo, y convertir las habilidades individuales en buenas prácticas del trabajo en grupo (compartir conocimiento). Además de la inspección y adaptación, los procesos empíricos poseen un tercer pilar muy importante, que es la transparencia de la información ante todos los integrantes del equipo de trabajo.

Ciclos de vida (Modelos de Proceso) y Procesos de Desarrollo de Software.

Ventajas y desventajas de c/u de los ciclos de vida. Criterios para elección de ciclos de vida en función de las necesidades del proyecto y las características del producto.

Criterios para la selección de un modelo de proceso

Es necesario evaluar los niveles de riesgo, la claridad y cantidad de requerimientos, los conocimientos técnicos con los que cuenta el equipo de desarrollo, los plazos que se tienen para ejecutar el proyecto y la posibilidad de poder validar los resultados del trabajo con el cliente, también el trabajo en equipo.

1. Si los requerimientos son claros y exhaustivos, en un contexto de baja incertidumbre en el que el cliente no requiere obtener el producto rápidamente, es posible aplicar el modelo secuencial.

2. Si la situación anterior no se presenta y el cliente desea obtener resultados lo antes posible, con las funciones principales del sistema, entonces sería conveniente la elección del modelo iterativo- incremental.

3. Si el objetivo es disminuir los riesgos presentes en el desarrollo de sistemas grandes y complejos, entonces el modelo recursivo puede ser una buena opción.

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.

Diferentes modelos de ciclo de vida

Code and Fix

Se desarrolla sin especificaciones o diseño y se lo modifica hasta que el cliente este satisfecho. Los cambios se realizan durante el mantenimiento, lo que es muy caro.

Este modelo tiene dos ventajas:

- No se dedica tiempo a planificar, documentar, hacer control de calidad o cualquiera actividad que no sea codificación pura. Es decir, se salta directamente desde un principio a la codificación y los progresos se ven inmediatamente.
- Requiere poca experiencia, cualquiera que sepa un lenguaje de programación puede familiarizarse con este modelo.

Puede ser útil para proyectos pequeños que se tiene la intención de desecharlos después de construirlos.

Modelo en cascada puro

Se sigue una secuencia de pasos ordenada y se realiza revisión al final de cada fase. Es conducida por la documentación con fases que no se superponen.

- Permite encontrar errores en etapas tempranas.
- Hay exceso de documentación y falta de resultados hasta el final.
- Se utilizan cuando la definición del producto es estable o los requerimientos de calidad dominan a los costos y tiempos.
- Funciona bien cuando se tiene un equipo técnicamente débil o con poca experiencia porque provee al proyecto una estructura que ayuda a minimizar el esfuerzo desperdiciado.
- Este modelo no es flexible, se requiere especificar completamente los requerimientos al comienzo y puede pasar mucho tiempo hasta que se tenga software funcionando. Olvidarse de algo puede ser un error muy costoso.

Modelo en cascada con fases solapadas

Hay un fuerte grado de solapamiento. La documentación es menor, pero es más difícil hacer el seguimiento de progreso.

Modelo de entrega por etapas

Se ven signos tangibles de progreso. Es útil cuando el cliente necesita funcionalidad de inmediato y las necesidades se modifican. Debe haber una planificación rigurosa para que funcione.

Modelo en cascada con retroalimentación

Es una variación del modelo clásico en que es posible volver a una etapa anterior antes de llegar al final, aunque es muy caro hacerlo.

Modelo en cascada con subproyectos

Es un modelo secuencial. Se avanza en cascada hasta el diseño arquitectónico, de ahí en más se avanza en partes dividiendo en Subproyectos.

Modelo en espiral

Busca minimizar los riesgos haciendo un análisis de riesgos y buscando alternativas al iniciar cada fase. Al final de cada fase se planifica la siguiente y se evalúa si se sigue adelante. Este modelo parte el proyecto en mini proyectos que permite manejar mejor los riesgos.

- Permite evaluar la factibilidad de un nuevo producto.
- En proyectos grandes la identificación de los riesgos puede costar más que el desarrollo.
- Es un modelo adecuado para proyectos de vida largos, que puede utilizar equipos diferentes en cada ciclo y muestra el progreso al final de este.
- Por lo general a medida que el costo incrementa, el riesgo disminuye. Mientras mas tiempo y dinero gastas, menos riesgos tomas, que es lo que se quiere en un proyecto de desarrollo rápido.
- Puede ser difícil definir objetivos e hitos que identifiquen que se está listo para avanzar a la siguiente capa del espiral.

Modelo evolutivo

Se fija el tiempo o el alcance mientras el otro se va modificando. Es útil cuando los requerimientos no son claros y se realizan entregas tempranas al cliente.

- El problema es que es un modelo recursivo que repite todas las tareas y depende de que los diseñadores desarrollen un sistema fácil de modificar.

Diseño para cronograma

No se asegura alcanzar la versión final, aunque si una entrega del producto para la fecha definida con los aspectos más importantes. Es útil cuando no hay seguridad sobre las capacidades de programación.

Diseño para herramientas

Consiste en incluir solo la funcionalidad soportada por las herramientas de software existentes. Se usa en proyectos muy sensibles al tiempo. Puede combinarse con otros ciclos de vida, pero no se podrán implementar todos los requerimientos.

Prototipación evolutiva

Se da cuando hay cambios rápidos de requerimientos y los clientes no se comprometen con los requisitos. Requiere menos documentación, pero no se pueden programar los release. Se desarrolla el concepto del sistema a medida que se avanza. Es útil cuando los desarrolladores no están seguros de que algoritmos o arquitectura optima utilizar.

Componentes de un Proyecto de Sistemas de Información.

Proyecto

Un proyecto de software es un esfuerzo temporal que requiere del acuerdo de un conjunto de especialidades y recursos para la creación de un producto, servicio o resultado único. Es una unidad organizativa para adaptar el marco teórico del proceso, dependiendo de las personas y recursos con los que se cuente.

Define el proceso y tareas que se van a realizar, el personal que se encargará de las actividades y los mecanismos que se implementarán para valorar riesgos, controlar el cambio y evaluar la calidad.

Características

Únicos: El resultado de un proyecto será único e irrepetible para cualquier otro proyecto, por más parecido o igual sean sus elementos componentes.

Todos los proyectos por más similares que sean tienen características propias que los hacen únicos, como por ejemplo el calendario (es decir, cronogramas diferentes), o presupuestos.

1. Orientado a objetivos:

No ambiguos: deben ser lo suficientemente claros para guiar el desarrollo del proyecto.

Medibles: es necesario medir el objetivo para determinar el avance sobre el proyecto.

Alcanzable: deben ser factibles de realizarse por el equipo (que el equipo pueda hacerlos).

Alcanzable: deben ser factibles de realizarse por el equipo (que el equipo pueda hacerlos).

Duración limitada de tiempo: Esto implica que un proyecto tiene un principio y un fin, liberando los recursos y personas. Cuando se alcanzan los objetivos del proyecto, este llega a su fin. Un ejemplo claro de lo que no es un proyecto es una línea de producción, ya que esta se ejecuta indefinidamente en el tiempo.

Conjunto de tareas interrelacionadas basadas en esfuerzo y recursos: Se definen las tareas, sus dependencias, los recursos y personas asignadas, permitiendo manejar la complejidad inherente de los sistemas.

Administración de proyecto de software

La administración de proyectos es la organización y coordinación de personas y recursos que permiten desarrollar un producto o servicio de acuerdo con el presupuesto, tiempo y funcionalidades acordadas con el cliente, asegurando que se entregue un producto o servicio de alta calidad. La administración de proyectos es una parte fundamental para que el proyecto sea exitoso. Esta disciplina tiene dos grandes actividades:

- Planificación.

- Monitoreo y control.

La administración incluye:

1. Identificar los requerimientos.
2. Establecer objetivos claros y alcanzables.
3. Adaptar las especificaciones, planes y el enfoque a los diferentes intereses de los involucrados (stakeholders).

Una buena gestión no puede garantizar el éxito del proyecto. Sin embargo, una mala gestión usualmente implica una falla en el proyecto: el software puede entregarse tarde, costar más de lo estimado originalmente o no cumplir las expectativas de los clientes. (esto se verá mejor explicado en la triple restricción).

Un proyecto planificado puede fracasar también, lo importante de la planificación es el acto de planificar, no el resultado. El acto de ponerme en el ejercicio de pensar tanto: riesgos, objetivos, alcances, estimaciones, etc.

Metas más importantes de la administración de proyectos:

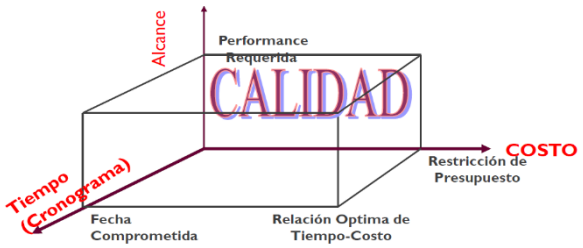
1. Entregar software al cliente en el tiempo acordado.
2. Mantener los costos dentro del presupuesto general.
3. Entregar software que cumpla con las expectativas del cliente.
4. Mantener un equipo de desarrollo óptimo y con buen funcionamiento.

Atributos particulares o complejidades en la administración de proyectos de software:

1. El producto es intangible: No poder ver ni tocar el producto que se obtiene al ejecutar el proyecto dificulta la medición del progreso del proyecto. Una construcción civil es visible y se puede evidenciar el progreso en función de las partes construidas.
2. Los proyectos de software suelen ser excepcionales: Los conocimientos aprendidos en un proyecto de software son difícilmente extrapolables a otro proyecto, como validación de riesgos, aplicación de nuevas tecnologías, etc. El factor tecnológico es una de las causas.
3. Los procesos de desarrollo son variables y específicos de la organización: A pesar del intento de estandarizar los procesos de desarrollo, la realidad es que no es posible predecir de manera confiable qué proceso es el adecuado para adoptar en el proyecto.

La triple restricción – enfoque tradicional

Las variables de restricción de un proyecto son 3: tiempo, costo y alcance. El balance de estos tres factores afecta directamente la calidad del proyecto, ya que un proyecto de alta calidad es aquel que entrega el producto, servicio o resultado requerido, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado. Es responsabilidad del Líder de proyecto balancear estas variables.



El concepto de la triple restricción hace referencia a que no es posible fijar de forma arbitraria a las 3 variables, sino que será posible fijar 2 y la tercera se ajustará a lo definido.

No se negocia la calidad, si no puedo modificar tiempo y/o costo, debido a que es como entregar un software sin probarlo. La calidad del producto que le entrego al cliente se ve afectada cuando no puedo equilibrar estos 3 factores.

Alcance – ¿Qué trata de alcanzar?

Son los requerimientos que se incluyen en el proyecto y definen la funcionalidad que tendrá el producto a entregar al cliente. En las metodologías tradicionales el alcance es lo primero que se determina y se deja fijo en función de las

necesidades del cliente, mientras que en ágil se va definiendo, refinando y adaptando en base a las retroalimentaciones del cliente. Esta variable es la que primero se negocia con el cliente.

Tiempo - ¿Cuánto tiempo me llevaría?

Define la fecha de calendario en la cual se compromete a finalizar el proyecto y entregarle el producto resultante al cliente. En las metodologías ágiles, el tiempo es fijo.

Costo - ¿Cuánto debería costar?

Define el costo de todos los recursos implicados en el desarrollo del proyecto. Esto incluye equipamiento, infraestructura, equipos, salarios, entre otros.

En la realidad, el costo y el tiempo del proyecto varían de forma directa con la definición del alcance del producto, es decir, el alcance es directamente proporcional al tiempo y al costo. Cuando el alcance aumenta (mayor cantidad de funcionalidades), el tiempo y dinero (costos) necesarios también deben aumentar, para lograr abordar un proyecto más grande. El problema en la aplicación de esta visión en enfoques tradicionales es que, al utilizar procesos definidos con ciclos de vida en cascada, los requerimientos se definen al comienzo del proyecto, por lo que cualquier modificación o cambio en el alcance del producto final, modifica el tiempo y presupuesto de este. Estos cambios, generan una baja en la calidad del producto final ya que, que se sacrifican funcionalidades o bien, se saltean etapas de testing, etc. debido a que el cliente siempre espera que se entregue lo pactado, respetando el presupuesto y tiempo de entrega.

Líder de Proyecto

En un enfoque tradicional, el líder de proyecto realiza la toma de decisiones en su totalidad sin la inclusión de la opinión del equipo. El líder asigna las tareas que deben realizar los integrantes del equipo, y además, maneja todas las relaciones con todos los stakeholders del proyecto (clientes, gerencias, contratistas, stakeholders en general).

Usualmente los profesionales técnicos no disponen de habilidades necesarias para tratar con la gente, por lo que es necesario una figura de líder de proyecto con las siguientes aptitudes:

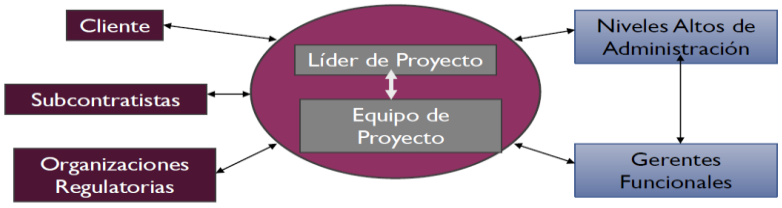
1. Motivación: habilidad para alentar al personal técnico a producir a su máxima capacidad.
2. Organización: habilidad para adaptar el proceso existente (o inventar nuevos), para lograr que el concepto inicial se transforme en el producto final.
3. Innovación: habilidad para alentar a las personas a crear y sentirse creativas.
4. Empatía.
5. Comunicación.
6. Liderazgo.

También cuenta con habilidades duras, relacionadas a los conocimientos del producto, técnicas y herramientas. Dado que el líder de proyecto tiene un enfoque basado en la resolución de problemas, también son características propias las siguientes:

1. Resolución de problemas: identifica conflictos técnicos y organizativos, estructura la solución o motiva a otros profesionales para que la desarrollen, aplica lecciones aprendidas en otros proyectos y adapta el proceso de resolución ante inconvenientes.
2. Identidad administrativa: tiene la confianza de asumir el control cuando es necesario.
3. Logro: recompensa las iniciativas y los logros, incentivando al equipo a correr los riesgos de manera controlada.
4. Influencia: define la estructura del equipo en función de la retroalimentación que él mismo suministra mediante palabras y gestos.

Entre las responsabilidades del líder de proyecto se encuentran, en las nuevas metodologías:

1. Definir el alcance del proyecto.
2. Identificar a los involucrados, recursos y presupuesto.
3. Detallar las tareas, estimar los tiempos y requerimientos.
4. Identificar y evaluar riesgos.
5. Preparar planes de contingencia.
6. Controlar hitos y participar en las revisiones de las fases del proyecto.
7. Administrar el proceso de control de cambios.
8. Producir reportes de estado.



Equipo de proyecto

Es un grupo de personas comprometidas con alcanzar un conjunto de objetivos de los cuales se sienten mutuamente responsables.

Entre sus características se encuentran:

1. Cuentan con conocimientos, habilidades técnicas, motivación, experiencias y diversas personalidades. Además, deben tener un sentimiento de mejora continua y aprendizaje mutuo.
2. Usualmente son un grupo pequeño, esto permiten lograr una comunicación efectiva entre los miembros del equipo.
3. Trabajan en forma conjunto con espíritu de grupo, convirtiéndose en un equipo cohesivo donde prima la sinergia de este. Para eso, se debe lograr un equipo maduro y estable, es decir, que no se agrega o cambia un integrante constantemente.
4. Sentido de responsabilidad como una unidad y se focalizan en el cumplimiento de las metas grupales.

En equipos cohesivos:

1. EL grupo establece sus propios estándares de calidad.
2. Los individuos aprenden de los demás y se apoyan mutuamente.
3. El conocimiento se comparte.
4. Prevalece la mejora continua.

Stakeholders

Son los interesados del proyecto, incluye el equipo de proyecto, el equipo de dirección, el líder de proyecto y el patrocinador.

Plan de proyecto

El plan de proyecto, (o también llamado plan de desarrollo de software en el ámbito del software), es un artefacto de gestión que cumple la función de “hoja de ruta” de un proyecto, en la cual se documentan todas las decisiones que se toman a lo largo del desarrollo; en términos de: el proceso, la división de tareas a realizar, asignación de responsabilidades, equipos de trabajo, calendario de desarrollo, actividades de soporte (planes de contingencia, mecanismos para control de cambios, evaluar calidad y valorar riesgos).

En un plan de proyecto se establece qué se va a hacer (alcance del proyecto), cuándo se va a hacer (calendario), cómo se va a hacer (actividades o tareas para cubrir el alcance) y quién lo va a hacer (responsables de cubrir las actividades). Responder a estas preguntas, implica las siguientes actividades:

1. Definición del alcance del proyecto.
2. Definición de proceso y ciclo de vida.

3. Estimación.
4. Gestión de riesgos.
5. Asignación de recursos.
6. Programación de proyectos.
7. Definición de métricas.
8. Definición de controles.

1. Definición del alcance del proyecto

El alcance de un proyecto, junto a su objetivo, responden al qué se está desarrollando. Es importante diferenciar el alcance de un producto y el del proyecto en sí. El alcance de un producto define qué funcionalidades debe realizar el software para satisfacer los requerimientos del cliente y se mide contra la Especificación de Requerimientos (ERS) (objetivos del producto), en cambio, el alcance del proyecto define todo el trabajo y solo el trabajo necesario que se debe realizar para cumplir con el desarrollo de este y poder entregar el producto o servicio con todas las características y funciones especificadas. El alcance del proyecto se mide contra el Plan de Proyecto (objetivos del proyecto).

Tanto el alcance como el objetivo de un proyecto puede ser modificado a lo largo del proyecto, pero esto no necesariamente tiene que cambiar el alcance del producto. Por otra parte, el alcance de un producto sí condiciona los alcances de un proyecto. (Puede ocurrir que no siempre sea así)

Es importante aclarar que el alcance del proyecto es menor que el alcance del producto. Ejemplos de alcances de un proyecto, pueden ser el realizar la toma de requerimientos, realizar Testing de determinados componentes, hacer el desarrollo del código de un componente, etc.

2. Definición del proceso y ciclo de vida

Al definir el proceso que se va a utilizar, se responde al cómo se va a desarrollar el proyecto. Es importante tener en cuenta la información del contexto del desarrollo: objetivos, alcances, personas y recursos disponibles, clientes, forma de trabajo de personas, etc. Por ejemplo, no es lo mismo utilizar un proceso y ciclos de vida para trabajos presenciales que remotos.

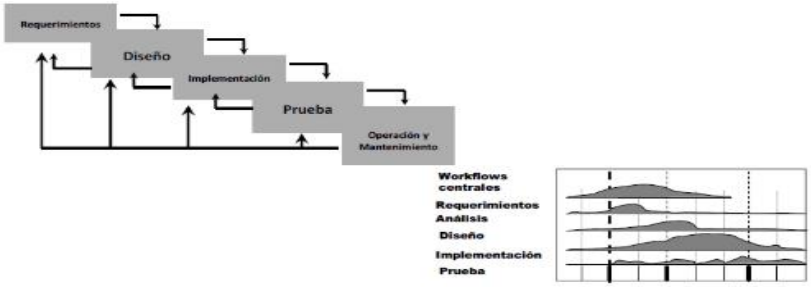
También se debe tener en cuenta la complejidad del software a desarrollar (alcances y objetivos) para así poder adaptar el proceso definido que se elija al proyecto en el cual se está trabajando.

El ciclo de vida del proyecto es lo que define cuánto de cada tarea se debe hacer y en qué momento del desarrollo del proyecto hacerlo.

El ciclo de vida de un proyecto define lo siguiente:

1. Qué trabajo técnico debería realizarse en cada fase.
2. Quién debería estar involucrado en cada fase.
3. Cómo controlar y aprobar cada fase.
4. Cómo deben generarse los entregables.
5. Cómo revisar, verificar y validar el producto.

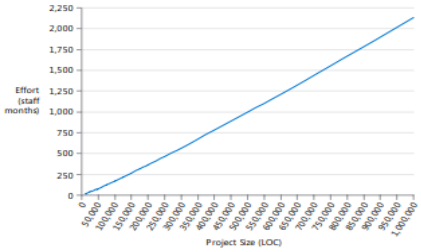
En metodologías tradicionales, se permite elegir cualquier tipo de ciclo de vida (cascada, iterativo e incremental, espiral, etc.), en cambio en metodologías ágiles esto no es posible (si o si tiene que ser iterativo e incremental).

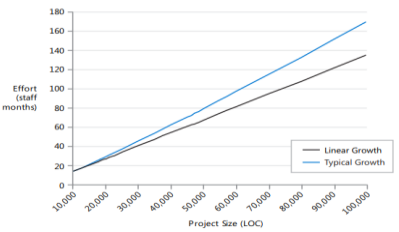


3. Estimación

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. La mayoría de las veces, el error de las estimaciones recae en estimar costos, tiempos y esfuerzo sin tener una estimación del tamaño, y luego en el transcurso del desarrollo, no variar este costo, tiempo y esfuerzo si varía el tamaño del producto (gran desventaja de la metodología tradicional → no responder a cambios de forma flexible). El tamaño de un producto se puede estimar en muchas escalas, por ejemplo: líneas de código, cantidad de requerimientos, puntos de función ajustados, etc.
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 personas lineales serán necesarias para construir el producto, previamente estimado su tamaño. Como dijimos anteriormente, el esfuerzo varía directamente con el tamaño de un proyecto, ya que, para un proyecto grande, es necesario tener más de una persona cumpliendo el mismo rol, o agregar roles diferentes. Cada rol, puede tener un valor de hora distinto o bien, se puede realizar una estimación en horas planas, en donde no se hace distinción de valor por rol. El esfuerzo es el tópico que más repercute en el costo del proyecto, abarcando casi el 80% de estos.





Además, el efecto de las habilidades personales de los integrantes del grupo define el esfuerzo final del proyecto, es decir dependiendo de la fuerza o debilidad en las habilidades de las personas, los resultados del esfuerzo proyecto pueden variar en una cantidad indicada (gráfico). Por ejemplo, un proyecto con los peores analistas de requisitos requeriría un 42% más de esfuerzo que el nominal, mientras que un proyecto con los mejores analistas requeriría un 29% menos de esfuerzo que el nominal.

3. **Calendario:** 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. Es necesario realizar esta estimación para poder darle una idea al cliente de cuánto tiempo se va a demorar la finalización del proyecto, y como en los demás aspectos, es necesario ya tener una buena estimación del tamaño y del esfuerzo que se requiere, ya que el tiempo de duración del proyecto es proporcional a su crecimiento.
4. **Costos:** de forma similar a la estimación de la duración del proyecto, la estimación de costos depende directamente del tamaño y del esfuerzo que se necesite para el desarrollo del software, por eso es lo último que se estima. De la misma manera, la estimación de costos es muy útil para darle una idea al cliente de cuál es el costo aproximado del proyecto.

4. Gestión de riesgos

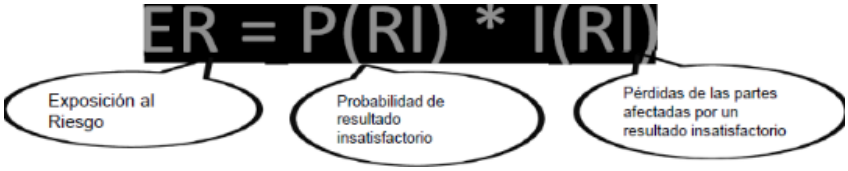
Un **riesgo** es la probabilidad de ocurrencia de una pérdida o daño que afecte de forma directa al proyecto y pueda comprometer el incumplimiento de su objetivo.

Es de suma importancia identificar y definir los riesgos a los que está atado un proyecto, para así poder analizarlos, y realizar una cuantificación de estos, teniendo en cuenta la exposición del proyecto ante los mismos, para así definir a cuáles se debe atender y sobre cuáles se debe hacer un seguimiento.

Los podemos clasificar en:

1. Riesgos del proyecto: aquellos que alteran el calendario o los recursos del proyecto. Un ejemplo de riesgo de proyecto es la renuncia de un diseñador experimentado.
2. Riesgos del producto: aquellos que afectan la calidad o el rendimiento del software a desarrollar. Un ejemplo de riesgo de producto es la falla que presenta un componente que se adquirió al no desempeñarse como se esperaba.
3. Riesgos organizacionales o de negocio: aquellos que afectan la calidad o el rendimiento del software a desarrollar. Un ejemplo de riesgo de producto es la falla que presenta un componente que se adquirió al no desempeñarse como se esperaba.
4. Riesgos organizacionales o de negocio: aquellos que afectan a la organización que desarrolla o que adquiere el software. Por ejemplo, un competidor que introduce un nuevo producto.

Para medir los riesgos se utiliza lo que se conoce como exposición al riesgo (probabilidad x impacto), la cual se obtiene como el producto entre la probabilidad de ocurrencia de la amenaza por el impacto que generaría si realmente ocurriera. Este valor permite organizar y gestionar los riesgos, y se debe hacer foco en reducir la exposición del riesgo.



Antes los riesgos identificados, es posible tomar 3 actitudes:

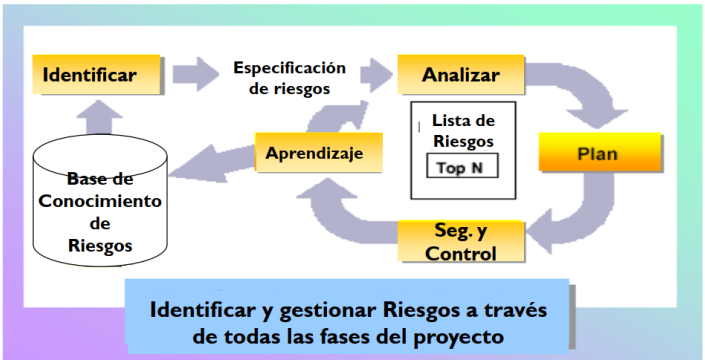
- Negación: hacer de cuenta que los riesgos no existen.
- Gestión Reactiva: esperar a que el riesgo ocurra y recién ahí ver que hacer.
- Gestión Proactiva: ir trabajando sobre el riesgo generado planes de mitigación y de contingencia.

Al tomar una actitud proactiva, la identificación y análisis de estos riesgos, permite definir para cuáles de esos riesgos identificados es importante definir planes de mitigación y contingencia. La Identificación y análisis permite reconocer los riesgos y analizar su importancia, luego los planes de mitigación permiten bajar la exposición del proyecto ante ese riesgo identificado y analizado. Por último, los planes de contingencia permiten actuar en caso de que un riesgo suceda, y poder volver a la normalidad lo antes posible.

Entre los mayores riesgos en el entorno de desarrollo de software se encuentran:

- Cambio de requerimientos en el medio de un desarrollo (volatilidad de requerimientos).
- Abandono de integrantes del equipo de desarrollo (alta demanda en mercado laboral).
- Falta de capacidad del equipo.
- El gran avance tecnológico, es decir, no tener en claro el funcionamiento de las últimas tecnologías.

Gestión del riesgo



1. **Identificación de riesgos:** se identifican los riesgos que suponen una mayor amenaza al proceso de ingeniería de software, al software a desarrollar o a la organización que lo desarrolla (al recibir los requerimientos).
2. **Análisis de riesgos:** para cada riesgo identificado se realiza un juicio acerca de la probabilidad y del impacto. No existe una forma certera de realizar esto. Se utilizan intervalos de probabilidad y clasificaciones de gravedad. El criterio dependerá de la experiencia de quien realice el análisis de riesgos.
3. **Planeación del riesgo:** para cada riesgo analizado, se definen estrategias para manejarlos. Estas estrategias consisten en considerar acciones a tomar para minimizar o evitar el impacto sobre el proyecto como consecuencia de la ocurrencia de la amenaza que representa el riesgo. Otras estrategias consisten en desarrollar planes de contingencia, el cual define un plan para enfrentar una situación controversial.
4. **Monitorización del riesgo y control:** proceso que permite determinar que las suposiciones acerca de los riesgos de producto, proceso y organización no han cambiado. Esto permite revalorar al riesgo en términos de posibles variaciones de su probabilidad e impacto. Este proceso se aplica en todas las etapas del proyecto.

5. Asignación de responsabilidades (o 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.

6. Programación de proyectos (calendarización)

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.

Generalmente la calendarización se realiza a través de un Diagrama de Gantt, realizado a través de alguna herramienta, como Microsoft Project.

7. 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. Además, es imprescindible que las métricas sean representativas

para el entorno, es decir, que representen un aumento en la información y beneficien la practicidad del desarrollo y seguimiento.
Las métricas del proyecto se consolidan para crear métricas de proceso que sean públicas para toda la organización del software.
A diferencia de las metodologías ágiles, acá se definen métricas que miden y exponen los avances. En cambio en ágil, el mayor indicador de avance de un producto, son los incrementos que se entregan en cada iteración.
Las métricas básicas para un proyecto de software se clasifican en:

- Tamaño del producto
- Esfuerzo
- Calendario (tiempo)
- Defectos
- Costos

8. 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.

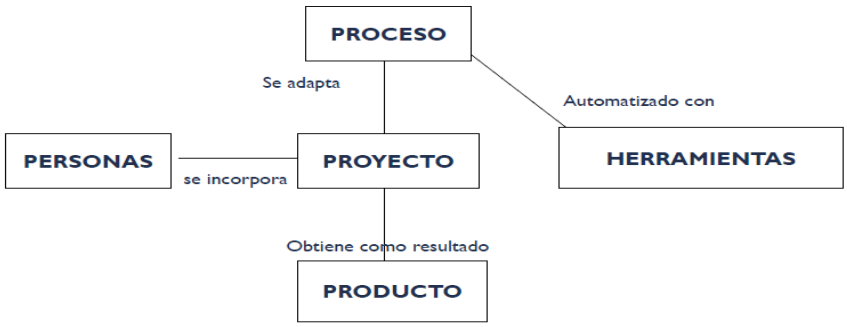
Planes de Soporte

- Planes de Testing.
- Planes de Mantenimiento.
- Planes de subcontrataciones.
- Planes de calidad.
- Planes de capacitaciones.
- Planes de iteraciones (si el ciclo de vida es iterativo/incremental).

Causas de fracasos de proyectos

- Fallas de toma de REQUERIMIENTOS (el 80%).
- Fallas al definir el problema.
- Planificar basado en datos insuficientes.
- La planificación la hizo el grupo de planificaciones.
- No hay seguimiento del plan de proyecto.
- Plan de proyecto pobre en detalles.
- Planificación de recursos inadecuada.
- Las estimaciones se basaron en “supuestos” sin consultar datos históricos.
- Nadie estaba a cargo.

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.

Para obtener como resultado un producto o servicio (ya sea SaaP o SaaS), es necesario que el proyecto adopte un proceso de desarrollo para poder definir qué actividades, métodos, prácticas y transformaciones se utilizarán, mediante las cuales se va a obtener el software. Dentro de cada proyecto de desarrollo de software, es necesario utilizar un proceso diferente dependiendo del tipo de producto que se desea desarrollar, ya que puede que la complejidad de este sea un determinante en la elección del proceso. Es por esto, que es necesario adaptar los procesos a cada uno de los proyectos en los cuales se trabaja, es decir, utilizar solo lo que verdaderamente hace falta para el desarrollo de este, ya que no existe un proceso ideal que sirva para cualquier tipo de proyectos. Es decir, el proceso es una definición teórica de lo que debería hacerse para tomar los requerimientos y transformarlos en un producto o servicio, guados por el objetivo de lo que se quiera construir.

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. Por ejemplo, el uso de un software de diseño para realizar el análisis y diagramación de clases.

No Silver Bullet

https://www.studocu.com/es-mx/document/universidad-anahuac/computacion-ii/no-silver-bullet/11916501

No hay balas de plata: Lo esencial y lo accidental en la ingeniería del software

- Contrastar progreso del hardware (algo tangible, construible, que se puede mejorar en su construcción), en comparación con el software (intangible, complejo inherentemente)
- Las balas de plata son aquellas soluciones que podrían hacer frente a la crisis del software: no existe un desarrollo de software que demuestre avances de progresión (simplicidad, fiabilidad y productividad) en la construcción del software en menos de una década.
- 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.

- **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.
- **Resolución 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.