# Unidad 1: Ingeniería de Software en Contexto

## 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.

Lo **más importante** en la ingeniería de software es la **gente** que produce y gestiona dicho software.

La evolución de la ingeniería de software ha sido impulsada por varios hitos:

1. **Crisis del Software (1960-1970)**: En la década de 1960, el rápido crecimiento de la demanda de software provocó lo que se denominó la "crisis del software". Proyectos complejos sufrían retrasos, sobrecostos y fallos debido a la falta de metodologías sistemáticas para el desarrollo. Esto llevó a la creación de técnicas más rigurosas y procesos de gestión.
2. **Nacimiento de la Ingeniería de Software (1968)**: El término "ingeniería de software" fue acuñado en la conferencia de la OTAN en 1968 para abordar los desafíos de la crisis del software. Se reconoció la necesidad de aplicar principios de ingeniería al desarrollo de software, formalizando su diseño, construcción y mantenimiento.
3. **Modelos de Ciclo de Vida (Décadas de 1970-1990)**: Modelos como el **ciclo de vida en cascada** fueron desarrollados para establecer una estructura en las fases de desarrollo. Este período también vio la adopción de metodologías más avanzadas como el **espiral** y el **desarrollo incremental**, que sentaron las bases de enfoques más ágiles.
4. **Metodologías Ágiles (1990-2000)**: El **Manifiesto Ágil** (2001) marcó un cambio hacia métodos de desarrollo más flexibles, centrados en la colaboración, la adaptación y la entrega continua de valor. Las metodologías ágiles desafiaron el enfoque rígido y secuencial de los modelos anteriores.
5. **Automatización y DevOps (2010-presente)**: En los últimos años, la ingeniería de software ha adoptado una mayor automatización, con herramientas para **integración continua** y **despliegue continuo**. El movimiento **DevOps** ha fomentado una colaboración más estrecha entre los equipos de desarrollo y operaciones.

## La Crisis del Software:

La **crisis del software** es un concepto que surge en los años 60 y 70 cuando se empezaron a evidenciar graves problemas en el desarrollo de software. A medida que los sistemas de software crecían en complejidad y magnitud, los proyectos comenzaron a sufrir **retrasos**, **sobrecostes** y **fallos** en cumplir con las expectativas iniciales.

Durante esa época, el desarrollo de software experimentó un auge debido al creciente uso de computadoras en diversos sectores, desde la industria hasta la ciencia.

Sin embargo, en ese momento, se utilizaban metodologías, que en su mayoría eran **ad hoc** y carecían de rigor estructural, no podían gestionar adecuadamente el desarrollo de sistemas cada vez más grandes y críticos. Entre los principales problemas que comenzaron a surgir se encuentran:

* **Retrasos en la entrega de proyectos:** Los desarrollos de software frecuentemente se extendían más allá de los plazos originalmente estimados, afectando tanto la economía de las empresas como la eficiencia operativa de los usuarios.
* **Sobrecostes:** Los presupuestos iniciales de los proyectos eran a menudo subestimados, lo que resultaba en incrementos significativos de los costos, que en algunos casos duplican o triplican los valores previstos.
* **Calidad deficiente:** El software que se entregaba a menudo estaba plagado de errores o defectos, lo que provocaba fallos frecuentes, mal rendimiento y una experiencia de usuario insatisfactoria.
* **Mantenibilidad limitada:** A medida que el software evoluciona, se hacía más difícil modificarlo o mantenerlo debido a la falta de documentación adecuada, el uso de técnicas no sistemáticas y la poca modularidad en su diseño.
* **No conformidad con los requisitos:** El software desarrollado muchas veces no lograba satisfacer completamente las necesidades del cliente o los requisitos inicialmente especificados.

El software nunca está a la altura, no hay software de suficiente calidad para satisfacer las demandas que hay. Cada vez hay más demanda, más necesidad y cada vez más software, por lo cual, siempre hay gente insatisfecha. Hay mucha competencia y los clientes tienen mucho más poder (si no te gusta una app la borras). La mayoría de veces buscamos soluciones técnicas a estos problemas cuando en realidad son problemas generados por los humanos en términos de comunicación, comprensión y manejo de expectativas.

(Estado actual): El avance de la **Ingeniería de Software** como disciplina ha permitido abordar muchos de estos desafíos a través de la introducción de metodologías, herramientas y técnicas más avanzadas.

Algunos de los avances que nos han permitido gestionar un poco la crisis son:

* Metodologías ágiles
* Automatización y herramientas avanzadas como por ejemplos los entornos integrados de desarrollo (IDE), y sistemas de gestión de versiones como GIT

**Problemáticas con el desarrollo de software**

* La versión final del producto no satisface las necesidades del cliente.
* No es fácil extenderlo y/o adaptarlo (agregar funcionalidad).
* Mala documentación.
* Mala calidad.
* Tiempo y costos excedidos a los del presupuesto.

**Motivos de productos de software exitosos**

* Involucramiento del usuario.
* Apoyo de la gerencia.
* Requerimientos claros.
* Planeamiento adecuado.
* Expectativas realistas.
* Hitos de control intermedios.
* RRHH competentes

**Causas de productos de software fallidos**

* Requerimientos incompletos.
* Falta de involucramiento del usuario.
* Falta de recursos.
* Expectativas poco realistas.
* Falta de apoyo de la gerencia.
* Requerimientos cambiantes.

### Ejemplos de softwares:

**Healthcare.gov (2013) (fracaso)**

El portal del gobierno de EE. UU. para la Ley de Cuidado de Salud a Bajo Precio (Affordable Care Act) sufrió un lanzamiento desastroso. El sitio web no soportó el tráfico esperado, lo que provocó fallos y caídas constantes. Los problemas se debieron a la falta de pruebas, coordinación inadecuada entre contratistas y un alcance mal definido.

**NHS National Programme for IT (Reino Unido, 2002-2011) (fracaso)**

Este ambicioso programa de digitalización del Servicio Nacional de Salud del Reino Unido, con un costo estimado de 12.700 millones de libras, fue cancelado después de casi una década. Las razones fueron la mala gestión del alcance, la falta de adopción por parte de los usuarios finales y la complejidad técnica subestimada.

**FBI Virtual Case File (2000-2005) (fracaso)**

El proyecto para modernizar los sistemas de gestión de casos del FBI fue abandonado después de gastar más de 100 millones de dólares. El fracaso se debió a una mala planificación, cambio constante de requisitos y problemas técnicos derivados de un enfoque de desarrollo inflexible.

**Google Search (1996 - Presente) (éxito)**

El motor de búsqueda de Google, iniciado como un proyecto académico, se ha convertido en uno de los proyectos de software más exitosos. Su éxito se debe a un enfoque centrado en el usuario, escalabilidad técnica, y una estrategia de mejora continua mediante la recolección de datos y retroalimentación en tiempo real.

**Amazon Web Services (2006 - Presente) (éxito)**

AWS ha sido un proyecto revolucionario en la nube, transformando la infraestructura de TI a nivel global. Su éxito radica en la flexibilidad de los servicios, la capacidad de crecimiento escalable, y la atención meticulosa a las necesidades del mercado, con una implementación continua y ágil.

**Microsoft Windows (Desde 1985) (éxito)**

A pesar de los altibajos en las versiones, el sistema operativo Windows ha sido uno de los proyectos de software más exitosos a lo largo de los años. Microsoft ha mantenido una estrategia centrada en la compatibilidad, facilidad de uso y constante evolución, lo que ha permitido que sea el sistema operativo dominante en el mundo.

## Disciplinas que conforman la ingeniería de software:

Diseño se enfoca en las disciplinas técnicas.

Disciplinas:

* **Técnicas** -> vinculada al producto de software. Nos referimos a actividades que aportan al desarrollo de software como producto. Requerimientos.
  + Análisis y desarrollo. -> La materia diseño se centra en este apartado.
  + Implementación.
  + Prueba.
  + Despliegue.
* **Gestión** -> vinculada al proyecto. Hacen referencia a actividades de planificación, monitoreo y control del proyecto de desarrollo de software.
  + Planificación.
  + Monitoreo y control de proyecto (también le dicen, seguimiento y supervisión).
* **Soporte o protectoras** -> son disciplinas transversales. Aseguramiento de calidad.
  + SCM: gestión de configuración de software.
  + Métricas.

## Ciclo de vida:

Es la serie de pasos a través de los cuales el producto o proyecto progresa. Es decir, que es una representación simplificada de un proceso, el cuál define elementos del proceso y el flujo del proceso.

Los productos tienen su ciclo de vida, al igual que los proyectos.

Los ciclos de vida iterativos pueden ser de duración fija o de alcance fijo. El enfoque del ciclo de vida iterativo e incremental del proceso unificado es de alcance fijo, mientras que el enfoque del ciclo de vida iterativo e incremental de scrum es de duración fija.

En el proceso unificado el ciclo de vida termina cuando se logran los casos de uso, se plantea una duración inicial, pero si no se logra en ese tiempo la iteración se alarga.

En scrum o agile en general, se establece una duración fija y se plantean algunas características, sino se logra en ese tiempo desarrollar esas características se entrega con menos, pero la duración no se alarga, no se negocia. Esto se denomina timebox, el tiempo no se modifica.

### Relación: 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.

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.

### Ciclos de vida de proyectos de software:

* Un ciclo de vida de un proyecto de software es una representación de un proceso. Grafica una descripción del proceso desde una perspectiva particular.
* Los modelos especifican:
  + Las fases de proceso.
  + El orden en el cual se llevan a cabo.

### Clasificación de los ciclos de vida:

* 1. Hay 3 tipos básicos de ciclos de vida para un desarrollo de software:
     + - * Secuencial.
     + Iterativo.
     + Recursivo.

#### Ciclo de vida 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.

Ventajas:

* **Claridad**: Al ser un proceso estructurado y predefinido, cada fase tiene un inicio y un final claro.
* **Facilidad de Gestión**: La gestión es más sencilla debido a que el progreso se mide por la finalización de fases específicas.
* **Adecuado para proyectos estables**: Es útil cuando los requisitos están bien definidos desde el principio y no cambiarán.
* **Documentación**: Genera una documentación detallada a lo largo del ciclo de vida, lo cual es beneficioso para la gestión y el mantenimiento.

Desventajas:

* **Rigidez**: No permite cambios significativos durante el proceso. Si hay un cambio en los requisitos, es costoso y difícil de implementar.
* **Retraso en la entrega de valor**: Los clientes no ven ningún producto funcional hasta que todas las fases estén completadas.
* **Riesgo elevado**: Si hay errores en fases tempranas, estos pueden no ser detectados hasta etapas tardías, lo que puede implicar grandes costos de corrección.

Criterios para elegir el ciclo secuencial:

* **Requisitos bien definidos**: Ideal para proyectos donde los requisitos son estables y bien entendidos desde el inicio.
* **Proyectos pequeños o medianos**: Funciona mejor en proyectos de tamaño reducido o mediano, donde las fases pueden completarse sin grandes problemas.
* **Entorno controlado**: Adecuado en proyectos donde no se espera mucha variabilidad o incertidumbre.
* **Documentación formal**: Si se requiere documentación extensa para cumplir con normativas o auditorías.

#### Ciclo de vida 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.

El sistema se desarrolla como una serie de versiones (incrementos) y cada una añade funcionalidades a la versión anterior.

Ventajas:

* **Entrega temprana de valor**: Los clientes pueden recibir versiones funcionales del producto en cada iteración, obteniendo valor desde etapas tempranas.
* **Flexibilidad ante cambios:** Permite ajustar los requisitos y funcionalidades en cada iteración según la retroalimentación recibida.
* **Reducción de riesgos**: Las iteraciones cortas permiten identificar problemas o errores rápidamente y corregirlos en las siguientes fases.
* **Mejora continua**: Se permite la revisión y mejora del producto durante todo el ciclo de vida.

Desventajas:

* **Requiere mayor gestión**: La gestión del proyecto es más compleja, ya que implica controlar múltiples ciclos y entregas continuas.
* **Sobrecarga en el equipo**: Cada iteración puede implicar la planificación, desarrollo, pruebas y entrega, lo que puede aumentar la carga de trabajo del equipo.
* **Entrega incompleta**: Si no se planifican adecuadamente los incrementos, puede haber versiones incompletas o poco funcionales del producto.

Criterios para elegir el ciclo iterativo/incremental:

* **Requisitos cambiantes**: Ideal cuando los requisitos no están completamente definidos al inicio y pueden evolucionar con el tiempo.
* **Proyectos grandes y complejos**: Adecuado para proyectos a gran escala, donde el producto puede desarrollarse en partes funcionales y entregarse gradualmente.
* **Necesidad de retroalimentación continua**: Se adapta bien cuando es importante obtener retroalimentación frecuente de los usuarios para mejorar el producto.
* **Minimización de riesgos**: Si es necesario mitigar riesgos mediante entregas tempranas y validación constante.

#### Ciclo de vida recursivo:

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

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. Se generan productos independientes de la implementación, que pueden ser reusables en sistemas de características similares.

Ventajas:

* **Resolución de problemas complejos**: Al dividir problemas en subproblemas, permite abordar sistemas complicados de manera más controlada.
* **Adaptabilidad**: Similar al iterativo, permite ajustes continuos y es adecuado para proyectos con alta incertidumbre.
* **Ciclo de mejora continua**: Los desarrollos se van ajustando y mejorando en cada paso recursivo, lo que asegura un crecimiento progresivo del producto.

Desventajas:

* **Complejidad elevada**: Requiere una alta coordinación entre equipos, ya que las actividades se repiten y pueden solaparse.
* **Difícil de gestionar**: La repetición constante puede hacer que la gestión del proyecto se vuelva caótica si no se organiza adecuadamente.
* **Dependencia de la experiencia del equipo**: Los equipos necesitan tener experiencia suficiente para trabajar de manera eficiente en ciclos repetitivos.

Criterios para elegir el ciclo recursivo:

* **Problemas mal definidos**: Ideal para proyectos donde no se entiende completamente el problema al inicio y se debe ir resolviendo de manera gradual.
* **Proyectos exploratorios**: Funciona bien cuando se trata de innovar o explorar nuevas tecnologías o soluciones.
* **Mejora continua del producto**: Se utiliza cuando es necesario refinar el producto constantemente para mejorar su funcionalidad o adaptarlo a cambios en el entorno.

### Influencia de los ciclos de vida en la administración de proyectos de software:

Los **ciclos de vida de desarrollo de software** influyen en la **administración de proyectos** al definir cómo se estructura, organiza y gestiona el progreso del proyecto. Estos modelos proporcionan un marco que guía las actividades del equipo y la toma de decisiones, impactando en aspectos clave como la **planificación**, **gestión de riesgos**, **recursos**, **costos**, y **calidad**.

Influencias generales:

* **Planificación**: El ciclo de vida elegido afecta cómo se planifican las fases del proyecto. Modelos más secuenciales, como el **cascada**, requieren una planificación completa desde el inicio, mientras que modelos más iterativos y ágiles permiten ajustes continuos a lo largo del proyecto.
* **Flexibilidad y adaptación**: Ciclos de vida más flexibles como los **iterativos** o **ágiles** permiten adaptarse rápidamente a cambios en los requisitos o condiciones externas. Esto facilita a los administradores ajustar la dirección del proyecto en función del feedback, mientras que en ciclos más rígidos, los cambios pueden generar altos costos y retrasos.
* **Control y monitoreo**: La administración de proyectos se ve influenciada por el nivel de control necesario. Modelos secuenciales permiten una supervisión estructurada de cada fase, mientras que modelos incrementales o ágiles requieren un monitoreo constante debido a las entregas frecuentes y ajustes de prioridades.
* **Gestión de riesgos**: Modelos como el **espiral** o los **iterativos** permiten una identificación y mitigación de riesgos más temprana y frecuente, ya que se revisan continuamente en cada ciclo. En modelos más rígidos, los riesgos suelen identificarse más adelante en el proceso, lo que puede aumentar los costos de corrección.
* **Entrega de valor**: Los modelos que dividen el proyecto en fases o incrementos permiten entregar valor de forma continua, beneficiando tanto al cliente como al equipo de gestión con retroalimentación temprana. Los ciclos más lineales retrasan la entrega de valor hasta las fases finales.

## Administración de proyectos:

* Administración de proyectos es la aplicación de conocimientos, habilidades, herramientas y técnicas 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 (stakeholders).

### La restricción triple:

* Objetivos de proyecto (alcance): ¿qué está el proyecto tratando de alcanzar?
  + Son los requerimientos del proyecto, es decir, los límites o el ámbito sobre el cual se va a desarrollar el mismo.
* Tiempo ¿cuánto tiempo debería llevar completarlo?
  + Es la especificación de las fechas para realizar las entregas que determinarán el avance del proyecto.
* Costos: ¿cuánto debería costar?
  + Son los recursos implicados en el desarrollo del proyecto.
* El balance de estos tres factores afecta directamente la calidad del proyecto.
  + Proyectos de alta calidad entregan el producto requerido, el servicio o resultado, satisfaciendo los objetivos en el tiempo estipulado y con el presupuesto planificado.
* Es responsabilidad del líder de proyecto balacear estos tres objetivos (que a menudo compiten entre ellos).



Lo ideal para tener una ganancia económica es que se tengamos la posibilidad de modificar 2 de las 3, lo más sano es que el tiempo y el costo lo manejemos nosotros, los desarrolladores, y el cliente maneje los alcances.

En los procesos definidos parte desde que los alcances ya estén plenamente definidos, pudiendo modificar el costo y el tiempo.

En los procesos agiles parte desde que hay una IDEA de lo que se quiere, se parte desde un equipo ya definido (sería los recursos fijos) y el tiempo es fijo (el scrint es fijo). Luego de tener los recursos y el tiempo fijos, se ponen de acuerdo (desarrolladores y cliente) para establecer los que se va a desarrollar para esa iteración (lo que se va a desarrollar se saca del producto backlog).

El problema que tienen los procesos agiles es planear el contrato con el cliente, porque no está definido ni el producto ni el tiempo para el momento en donde se hace el contrato inicial. Los clientes no están acostumbrados a esta clase de contratos, ya que es un producto incremental.

## Procesos definidos vs procesos empíricos:

Existen dos tipos de procesos:

* Procesos definidos:
* Procesos empíricos:

El proceso se adapta al proyecto.

### Proceso definido:

El proceso definido es un proceso en el que están bien definidos un conjunto de actividades a seguir. Dados una serie de inputs, un proceso definido proceso el mismo output cada vez basado en la repetición y la naturaleza predictiva.

Un **proceso definido** es aquel en el que todas las fases y actividades están claramente especificadas, documentadas y repetibles. Estos procesos se basan en la idea de que se puede predecir y controlar cada paso del ciclo de vida del desarrollo de software.

Debido a la claridad en los pasos, el comportamiento del proceso es predecible. El progreso es medible y los resultados son consistentes siempre que se sigan las instrucciones.

Es adecuado para proyectos donde los requisitos son estables y no cambian mucho durante el desarrollo.

Se requiere una documentación exhaustiva en cada etapa, lo que asegura que todos los pasos sean seguidos de manera correcta y permite auditorías precisas.

Hay un contexto organización, que define la forma de trabajar, y espera que los proyectos que se vayan a llevar adelante sigan con esta forma de trabajar.

Hay una secuencia definida de pasos, se van adaptando las necesidades de cada proyecto en particular.

Tiene una retroalimentación más tardía.

Lo que buscan es poder repetibilidad, es poder estimar nuevos proyectos a partir de proyectos pasados. Ejemplos, PDU/RUP.

**Ventajas:**

* Estabilidad y control para proyectos con requisitos claros y poco cambiantes.
* Buena para proyectos grandes y de larga duración donde el seguimiento estricto es necesario.

**Desventajas:**

* Menos flexible ante cambios en los requisitos.
* Puede ser ineficiente si los requisitos cambian o si se presentan imprevistos.

### Procesos empíricos:

Se basan en la experiencia, pero plantean una forma de generar esa experiencia.

Un **proceso empírico** se basa en la experiencia directa, adaptabilidad y revisión constante. No se busca definir todos los pasos por adelantado porque se reconoce que es difícil o imposible predecir con precisión cómo se desarrollará el proyecto a lo largo del tiempo.

Sostiene que la experiencia no es extrapolable, es decir, la experiencia de uno no se transmite a los demás.

El enfoque empírico acepta que los requisitos pueden cambiar a lo largo del proyecto y, por lo tanto, permite flexibilidad.

El progreso se revisa de manera continua, y los resultados se ajustan según las observaciones y la retroalimentación.

Se utiliza cuando el proyecto tiene muchas incertidumbres o cuando los requisitos no están claros desde el principio.

La colaboración entre el equipo y con los clientes es crucial para ajustar el desarrollo a medida que se avanza.

Plantean que la mejor forma de tomar decisiones es que la realice el que va a hacer el proyecto. El equipo define como cree que es la mejor forma de construir software.

Los equipos tienen que funcionar autónomamente. Aprender de la experiencia, la experiencia obtenida por el equipo.

El empirismo trabaja con 3 pilares que son los que determinan que esto pueda funcionar, estos pilares son los que toman el agilismo:

* Inspección: paro lo que se está haciendo y reviso. Se inspecciona el proceso (se hace con el equipo) y el producto (se hace con el cliente).
* Adaptación: adaptarse a los cambios detectados en la inspección.
* Transparencia: la información debe ser visible y disponible para todas las personas implicadas en el proceso.

En la inspección y adaptación es donde se da la iteración, la realimentación es la experiencia.

**Ventajas:**

* Flexibilidad ante cambios en los requisitos.
* Capacidad de entregar valor continuo y obtener retroalimentación rápida.
* Ideal para entornos complejos y proyectos con alta incertidumbre.

**Desventajas**:

* Puede ser difícil de gestionar en grandes proyectos sin la experiencia adecuada.
* El riesgo de desvío es mayor si no se gestionan adecuadamente las expectativas.

### Diferencias entre el proceso empírico y definido:

|  |  |  |
| --- | --- | --- |
| Aspecto | Proceso Definido | Proceso Empírico |
| Estructura | Fija, basada en un plan detallado. | Flexible, basada en retroalimentación y ajustes. |
| Previsibilidad | Alta si no hay cambios en los requisitos. | Baja, depende de la retroalimentación constante. |
| Adaptabilidad | |  | | --- | |  |  |  | | --- | | Baja, difícil responder a cambios rápidos. | | Alta, fácil de adaptarse a cambios y ajustes. |
| Documentación | Extensa y detallada. | Suficiente para comunicar el estado, pero no exhaustiva. |
| Riesgo | Bajo (si el entorno es estable). | Alto (requiere buena gestión y comunicación). |
| Ideal para | Proyectos con requisitos claros y estables. | Proyectos con alta incertidumbre y requisitos cambiantes. |

La elección entre un proceso empírico o definido depende de la naturaleza del proyecto. Un enfoque definido es adecuado para proyectos bien conocidos y predecibles, mientras que un enfoque empírico es ideal para proyectos complejos y dinámicos donde los requisitos cambian constantemente.

## Componentes de un Proyecto de Sistemas de Información:

* **Hardware:** equipo físico utilizado para procesar y almacenar datos.
* **Software** y los procedimientos para transformar y extraer información.
* Los **datos** que representan las actividades de la empresa.
* La **red** que permite compartir recursos entre dispositivos.
* Las **personas** que desarrollan, mantienen y utilizan el sistema.

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

La administración efectiva de un proyecto de software se enfoca en las 4 P: **personal, producto, proceso** y **proyecto**.

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



* El Proyecto es el medio por el cual organizamos los recursos, asignar tareas y es el medio por el cual se entrega el producto de software al cliente.
  + Una vez definido el modelo de proceso en el contexto del proyecto, es posible realizar la planificación del proyecto. El proyecto es la unidad de gestión de recursos para obtener un producto o servicio determinado.
* El Producto es el resultado del proyecto y los procesos, es lo que se entrega al cliente, en este caso, sería el software funcionando.
  + 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 cómo se van a lograr. El ámbito del producto define las funciones y comportamientos que caracterizan al producto que utilizara cliente.
* Las Personas son el factor más importante en el software ya que determinan el éxito del proyecto de software.
* El Proceso es una descripción sobre lo que se debe hacer para construir el software. El proceso se adapta al proyecto.
  + El proceso proporciona un marco conceptual en donde se establece un plan completo para el desarrollo del software. Define cómo se va a desarrollar el software, estableciendo un conjunto de actividades.
  + 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.
* Las Herramientas nos ayudan a automatizar los procesos para que sean más eficientes y productivos.

Software es un conjunto de: programas, procedimientos, reglas, documentación y datos.

Software es conocimiento o información que se presenta en distintos niveles de abstracción.

La información tiene que estar estructurada con propiedades lógicas y funcionales. Además, es creada y mantenida en varias formas y representaciones. A su vez, es confeccionada para ser procesada por computadora en su estado más desarrollado.

### Proyecto:

Un proyecto es una unidad de gestión, que toma recursos y el tiempo con el objetivo de obtener un producto o un servicio.

**Características:**

* El resultado debe ser único.
* Tiene una fecha de inicio y una fecha de fin identificadas.
* Elaboración gradual: el proceso es el que hace la descomposición del proyecto.
* Se trabaja con tareas interrelacionadas.
* **Orientación a objetivos:** 
  + Los proyectos están dirigidos a obtener resultados y ellos refleja a través de objetivos.
  + Los objetivos guían al proyecto.
  + Los objetivos no deben ser ambiguos.
  + Un objetivo claro no alcanza, debe ser también alcanzable.
* **Duración limitada:** 
  + Los proyectos son temporarios, cuando se alcanza el/los objetivo/s, el proyecto termina.
  + Una línea de producción no es un proyecto.
* **Tareas interrelacionadas basadas en esfuerzos y recursos:** Complejidad sistémica de los problemas.
* **Son únicos:** Todos los proyectos por similares que sean tienen características que los hacen únicos.

#### Rol del líder de proyecto/equipo:

Encargado de definir el alcance del proyecto, detallar las tareas estimando tiempo y requerimientos, identificar recursos y definir presupuestos, identificar y evaluar riesgos, preparar planes de contingencia, participar en las revisiones de las fases del proyectos.

#### ¿Qué es un equipo de proyecto?

Un grupo de personas comprometidas en alcanzar un conjunto de objetivos de los cuales se sienten mutuamente responsables.

**Características de un equipo de proyecto:**

* Diversos conocimientos y habilidades.
* Posibilidad de trabajar juntos efectivamente/desarrollar sinergia.
* Usualmente es un grupo pequeño.
* Tienen sentido de responsabilidad como una unidad.

#### 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:

* Un plan es a un proyecto lo que una hoja de ruta a un viaje.
* El plan de proyecto documenta:
  + ¿qué es lo que hacemos?
  + ¿cuándo lo hacemos?
  + ¿cómo lo hacemos?
  + ¿quién lo va a hacer?

Objetivos del proyecto deben ser claros y alcanzables.

El plan de proyecto consta de:

1. Objetivo del proyecto.
2. Alcance del proyecto.
3. Equipo de proyecto.
4. Proceso y ciclo de vida.
5. Estimaciones.
6. Riesgos.
7. Recursos.
8. Programación/calendarización.
9. Métricas.
10. Informes y reuniones.

Estas 2 ultimas (9 y 10) es la planificación del seguimiento.

#### ¿Qué implica la planificación de proyectos de software?

* Definición del alcance del proyecto.
* Definición de proceso y ciclo de vida.
* Estimación.
* Gestión de riesgos.
* Asignación de recursos.
* Programación de proyectos.
* Definición de controles.
* Definición de métricas.

#### Definición del alcance:

* **Alcance del producto:** Son todas las características que pueden incluirse en un producto o servicio.
* **Alcance del proyecto:** Es todo el trabajo y solo el trabajo que debe hacerse para entregar el producto o servicio con todas las características y funciones especificadas.

Todo el trabajo y solo el trabajo que tengo que hacer para cumplir con el objetivo.

#### ¿Cómo se mide el alcance?

* El cumplimiento del alcance del proyecto:
  + Se mide contra el plan de proyecto (o plan de desarrollo de software).
* El cumplimiento del alcance del producto:
  + Se mide contra la especificación de requerimientos.