Tema 1. Gestión de Proyectos

Introducción

Una buena gestión de un proyecto es una condición necesaria para aumentar la probabilidad de éxito.
Esto quiere decir que se cumplen los **estándares** establecidos, se sigue la **agenda** prevista, se garantiza la calidad requerida y **no se sobrepasa el presupuesto**.
El responsable de que así se cumpla será el gestor que planificará el proceso de desarrollo y hará un seguimiento del trabajo.

Nuestros proyectos tienen una especial característica y es que generan un producto muy singular:

1. Un producto intangible como es el software, al que medirle la calidad es complicado.
2. La no existencia de estandarización del proceso de desarrollo del mismo.
3. falta de unicidad que permita comprender proyectos de grandes sistemas software desarrollados por empresas diferentes.

Así pues para lograr una buena gestión del proyecto tenemos tres grandes puntos a tener en cuenta:

1. Personal

Es necesario atraer, mostrar, acumular, desplegar y retener el talento en nuestra empresa para garantizarnos que nuestros recursos, que son los trabajadores son los mejores que podemos tenr y que pueden dar el máximo por el proyecto. Son clave, pues, los siguientes áreas:

1. Reclutamiento
2. Entrenamiento
3. Selección
4. Gestión de rendimiento
5. Retribución
6. Desarrollo de la carrera
7. Diseño de la organización y del trabajo
8. Desarrollo cultural
9. Espíritu de equipo

Dentro de la parte del personal tenemos como participantes:

1. **Usuarios finales:** Los que usaran el sistema desarrollado
2. **Clientes:** Quienes financian y pagan por el software, que no tienen porque ser los usuarios finales.
3. **Profesionales:** Los trabajadores o componentes del equipo. Pertenecen a diferentes disciplinas y categorias profesionales, con distinta formación.
4. **Gestores técnicos:** Con conocimientos para **planificar, motivar, organizar y controlar** a los profesionales. Es necesario tener una formación adecuada para ello.
5. **Gestores superiores:** Son gestores pero dedicados principalmente a encargos a nivel de empresa y comercial.

El jefe del grupo (suele ser un gestor técnico) tendrá una serie de habilidades como son:

- Motivar
- Incentivar la creatividad
- Crear un equipo cohesionado
- Moldear procesos (saber resolver problemas)
- Gestionar
- Incentivar logros

El equipo de trabajo podemos tener más trabajadores que tareas o al contrario. Sin embargo, hoy en dia se suele trabajar en equipos, donde no se asigna una tarea a un individuo, si no que se asigna a un grupo de individuos.
Estos equipos pueden presentar 3 estructuras diferentes:

1. **Descentralizada Democrática (DD):** En Este tipo de equipos, las decisiones se toman por consenso, y existe comunicación horizontal. No hay roles fijados, pero suele aparecer un líder de forma natural. Pueden cambiar según la tarea que está siendo abarcada por el equipo.
2. **Descentralizada Controlada (DC):** Hay un jefe principal y otros secundario. Se dan soluciones en grupo pero se implementan por subgrupos repartidos por el jefe. Comunicación horizontal (entre los grupos y sus individuos) y vertical (entre jefes de grupos al jefe superior).
3. **Centralizada Controlada (CC):** Existe una jerarquia bien definida, siendo el jefe el encargado de dar solución a los problemas de alto mal. Así la comunicación es entre jefe y equipo de forma vertical.

No existe una estructura única e ideal para el equipo que funcione con todos los proyectos.

Depende de 7 factores:

- Dificultad
- Tamaño
- Duración o tiempo de vida del equipo
- Grado de modularización del problema
- Calidad y fiabilidad requerida
- Rigidez del tiempo de entrega
- Grado de comunicación

DD —> **Dif:** Alta **Tam:** Pequeño **Dur:** Largo **Mod:** Baja **Fiab:** Alta **Ent:** Flexible **Com:** Alta
DC —> **Dif:**Baja **Tam:**Grande **Dur:**Corto **Mod:**Alta **Fiab:** Alta **Ent:** Flexible **Com:**Pequeña
CC —> **Dif:**Pequeña **Tam:**Grande **Dur:**Corto **Mod:**Alta **Fiab:**Baja **Ent:**Estricta **Com:**Pequeña

Problema

Es muy frecuente que cliente y desarrollador no se entiendan, y la solución aportada por el desarrollador (El sistema software) acabe siendo algo diferente y que no aporte solución al problema que se especifica por parte del cliente. Es necesario tomar unas estimaciones cuantitativas:

1. Analizar el ámbito del software, con sus objetivos, funciones y rendimiento.
2. Descomponer el problema o funcionalidades y requisitos para intentar darle solución.

Proceso

Podríamos definirlo como, la representación abstracta para perfilar desde un enfoque concreto el desarrollo de software aunque no representa exactamente como desarrollarlo.

El modelo de proceso de desarrollo software debe ser seleccionado de forma apropiada para el equipo del proyecto y que se adapte también al problema.
Existen varios modelos de proceso, los más comunes:

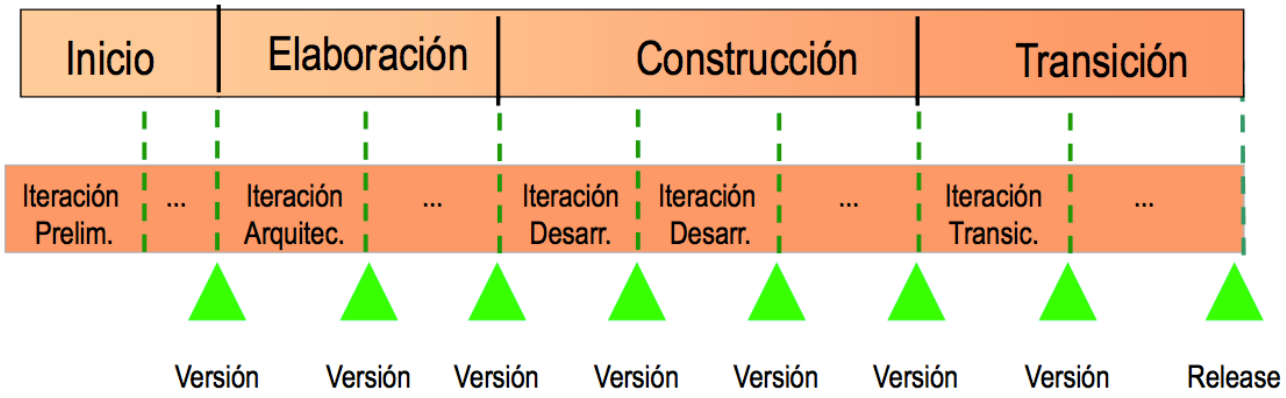
Secuenciales: Codificar y corregir, modelo en cascada, prototipos, DRA
Evolutivos: Desarrollo incremental, en espiral, UP (el que nos interesa)

Modelo UP (Unified Process)

El proceso unificado de desarrollo consta de 4 fases:

1. Inicio: Define ámbito del proyecto, casos de uso.
2. Elaboración: Plan del proyecto, estimaciones, diseño básico.
3. Construcción: Se implementa en base a iteraciones.
4. Transición: Entregar (se realizan pruebas beta).

Se divide en iteraciones donde a su fin siempre se han de **generar artefactos** nuevos, pero no son entregas para el usuario. Se trata de versiones internas. Al usuario se le entrega la **Release** al final de ciclo. Podemos tener las iteraciones que queramos pero al menos una por fase.



Una iteración es una secuencia de actividades con un plan establecido y un criterio de evaluación.

Finalmente, las actividades del gestor no acaban aquí, también tenemos que:

- Medir la calidad del producto
- Evaluación de riesgos
- Métricas
- Estimación de costes
- Confección de agendas
- Comunicación con el cliente
- Personal
- Otros recursos
- Monitorización del proyecto

Tema 2. Estimación de costes

La estimación de costes consiste en predecir los recursos no solo materiales si no que también **temporales, humanos y materiales** que necesitamos para llevar a cabo el proceso de desarrollo del software. No es pues lo mismo que el presupuesto. Fundamentalmente intentaremos resolver cuanto esfuerzo, tiempo y coste total suponen completar una actividad.

Principalmente hemos de tener en cuenta los costes de esfuerzo que son los sueldos de los ingenieros en el proyecto, juntos a gastos como seguridad social, seguros, viajes, alquiler del local, calefacción, etc.. Al coste de la aplicación también afectará otros factores como:

1. **Oportunidad de mercado**: Una empresa podría considerar lanzar un producto a muy bajo precio para ganar cuota de mercado. Ademas obtendría una experiencia para desarrollar futuros productos.
2. **Incertidumbre en al estimación de costes**: Ya que tenemos nuestras dudas sobre la estimación podríamos aumentar el precio para cubrir posibles futuros riesgos.
3. **Términos contractuales**: Vender el software con los derechos quedandonoslos nosotros, venderlo junto a los códigos fuente, no quedarnos nada…
4. **Volatilidad de los requerimientos:** Si sabemos que estos cambiaran mucho podemos reducir el precio del sistema para ganar el contrato para luego los cambios se cobrarían por mas precio.
5. **Salud financiera:** Sería mejor reducir el precio a tener que cerrar por no conseguir contratos para cubrir nuestros gastos. Es aconsejable evitar financiarnos con bancos y buscar autofinanciación o otros medios, pero no repercutir en nuestros beneficios.

Productividad

La productividad es una medida de la velocidad a la que los ingenieros producen el software y la documentación que lo acompaña. Su estimación es necesaria para estimar el proyecto así como su evaluación, al incorporar herramientas que pueden afectarla.

$$productividad = \frac{(atributo\ del\ software)}{esfuerzo\ total}$$

Atributos del software: Estos atributos pueden estar relacionados con el tamaño del software o con su funcionalidad. Estas son medidas basadas en volumen/tiempo.

- Tamaño
  - **LOC (lines of code)**

Muy sencilla de contabilizar. Propuesta desde los inicios cuando se programaba en tarjetas perforadas. Actualmente no es muy utilizada ya que no funcionan muy bien ya que los nuevos lenguajes de programación pueden contener mas de una instrucción por linea, o varias lineas para una instrucción. Es decir a medida que ha ido aumentando la expresividad de los lenguajes, la productividad se ve afectada negativamente en mayor medida. Además no se pueden comparar distintos lenguajes de programación, ya que nos llevaría a errores de medición.

	Analysis	Design	Coding	Testing	Documentation
Assembly code	3 weeks	5 weeks	8 weeks	10 weeks	2 weeks
High-level language	3 weeks	5 weeks	8 weeks	6 weeks	2 weeks

	Size	Effort	Productivity
Assembly code	5000 lines	28 weeks	714 lines/month
High-level language	1500 lines	20 weeks	300 lines/month

- Funcionalidad
  - **Puntos función**

    Dependen de 4 características:

    1. Entradas y salidas externas.
    2. Interfaces externas.
    3. Interacciones de usuario.
    4. Ficheros usados por el sistema.

    Se le asigna un peso a cada uno de ellos, incluyendo varios pesos dentro de la misma categoría para mayor ponderación. Se suman estos puntos y se multiplican por sus pesos, y finalmente se multiplican por el factor de complejidad.

$$Puntos\ de\ función = \sum_{\|}^{\|} (contador \cdot peso) \cdot \mathrm{C}$$

El factor de complejidad es C= 0,65 + 0,01* N, siendo N el grado de influencia de factores externos.

Las principales ventajas que presenta los puntos función son:

1. Son independientes del lenguaje de programación
2. Se puede calcular a partir de la especificación
3. Usa información del dominio del problema
4. Mas facil a la hora de reusar componentes
5. Mas orientado a objetos.

El principal inconveniente —> Son muy subjetivos, depende del estimador.

  - **Puntos objeto**

Otra alternativa más objetiva, que tiene en cuenta:

1. Pantallas visualizadas.
2. Informes que se producen
3. Módulos 3GL que deben desarrollarse para el 4GL.

También es posible utilizarlo en etapas muy tempranas, sin necesidad de tener el código, incluso de forma mas sencilla que los puntos de función. Así pues se puede estimar en etapas todavía mas tempranas. Son mas fáciles entonces que los puntos de función.

Ambos, puntos de función y de objetos, pueden convertirse a lineas de códigos utilizando unas constantes AVC, que varían entre lenguajes y dan información sobre la cantidad de lineas necesarias para implementar alguno punto.

$$LOC = AVC \cdot Número\ de\ puntos$$

Como conclusión…

La productividad se puede ver afectada por:

1. Experiencia de los desarrolladores
2. Calidad del proceso
3. Tamaño del proyecto
4. Tecnologias de soporte
5. Entorno de trabajo

Hay que recalcar que estas tecnicas para medir la productividad son engañosas pues no tienen en cuenta la calidad del producto desarrollado. Se puede incrementar la productividad y parecer que esto son mejoras, pero podríamos estar sacrificando la calidad. Estas métricas solo deberían servirnos como guias.

Técnicas de estimación

Estimar el esfuerzo requerido para desarrollar el sistema software no es sencillo ni es exacto, pero la experiencia ira haciendo que seamos cada vez mas precisos aunque siembro conservando un margen de error.

Las siguientes técnicas son aplicables cuando tenemos un documento con al especificación de los requerimientos del sistema:

1. **Modelado algorítmico de costes:** Se desarrolla un modelo empleando información histórica. Utilizando medidas cuantitativas de los atributos del sistema se realizan una predicción del esfuerzo. Es recomendable hacer 3 estimaciones (peor, esperado, mejor). Se aplica la fórmula:

$$esfuerzo = A \cdot Tamaño^{B} \cdot M$$

(Donde A es una constante en COCOMA que vale 2.5, B es un valor entre [1.1, 1.24] que según la novedad, flexibilidad, gestión de riesgos y madurez va variando, y M contiene 7 atributos del sistema contificados)

2. **Juicio Experto:** Expertos en el proceso de desarrollo elegido y el dominio de la aplicación llegan a una estimación por consenso. Si los estimadores son los correctos y buenos, se llega a una muy buena estimación con un error muy bajo. La principal ventaja que presenta es que es relativamente barato.
3. **Estimación por analogía:** Requiere del mantenimiento de una base de datos con información de proyectos anteriores. Intentamos estimar por semejanza con la experiencia de proyectos pasados. Puede ser muy preciso sis e dispone de dichos datos.
4. **Ley de Parkinson:** Costes en función por los recursos disponibles y no por objetivos. No se realizan estimaciones abultadas, pero no se suelen terminar los proyectos.
5. **Pricing to win:** Dependemos del presupuesto del cliente y eso es lo que le cobraremos, no dependiendo de la funcionalidad del sistema que seria ajustada luego. Con esta metodologia se suele conseguir el contrato pero no resulta en sistemas que cumplan con lo solicitado. Sin embargo se suele emplear cuando empezamos y el cliente no tiene claro lo que quiere para realizar el primer presupuesto.

La principal idea es combinar técnicas. Si varían mucho es que nos falta información sobre el sistema y recursos para estimar bien.

Cualquiera de estas aproximaciones puede abordarse desde el punto ascendente o descendente:

- **Estimación descendente:** Empieza por el sistema en general y tiene en cuenta la integración, gestiones de configuración y documentación. Puede infraestimar los costes a niveles técnicos de bajo nivel difíciles de resolver.
- **Estimación ascendente:** Empieza a nivel de modulo y es bastante exacto en su estimación si se ha diseñado el sistema en detalle. Puede infraestimar los costes de integración y documentación.

Tema 3. Planificación De Proyectos

La planificación del proyecto es una de las actividades de gestión que consume mas tiempo y una de las mas importantes. La elaboración del plan es ademas un proceso que se realiza de forma continuada durante todo el desarrollo del proyecto. Se deben revisar siempre que se tiene nueva información.
A parte del plan de proyecto existen otros planes, que describen:

1. **Plan de calidad:** Procedimientos o estándares a utilizar en el proyecto.
2. **Plan de validación:**Enfoque, recursos y programación utilizados para validar el sistema.
3. **Plan de gestión de configuraciones:** Procedimiento para al gestión de configuraciones y las estructuras a utilizar.
4. **Plan de mantenimiento:** Requerimientos para el mantenimiento del sistema, junto a los costes y esfuerzos del mismo.
5. **Plan de gestión de personal:** Desarrollo de sus habilidades y experiencias.

Planificar consiste en decidir, **que** hay que hacer, **como** hacerlo, **cuando** hacerlo y **quien** lo hace. Para ello, primero se establecen las restricciones del proyecto, se hacen evaluaciones iniciales de las mismas, se definen los hitos y productos a entregar. Entonces se entra en un ciclo. Ver imagen.

```
ALGORITMO:
Establecer restricciones proy.
Hacer evaluaciones iniciales
Definir hitos y entregas
Mientras proy no acaba hacer
        Confeccionar agenda
        Inciar trabajos agenda
        Esperar
        Revisar el progreso
        Revisar estimaciones
        Actualizar agenda
        Re-negociar restricciones
        Solucionar posibles problemas
Fin mientras
```

Este plan se divide en 7 etapas:

1. **Introducción:** Describir objetivos del proyecto y exponer restricciones.
2. **Organización del proyecto:** Describe el equipo de desarrollo
3. **Análisis de riesgos:** Gestión de riesgos
4. **Requerimientos HW y SW:** Describe ambos, lo que se va a emplear.
5. **Estructurar actividades del proyecto:** División del trabajo, marca los hitos y productos a entregar…
6. **Agenda del proyecto:** Dependencias entre actividades, estimación, recursos…
7. **Mecanismos de monitorización y control:**Gestión de informes, supervisión.

Las actividades se organizan de forma que produzcan salidas tangibles que permitan un control del progreso por parte del gestor. Así pues se marcan los **hitos o minestones** al final de las actividades. Las **entregas o deliveriables** son hitos cuyos resultados se entregan al cliente. El modelo en cascada marca bien estos hitos. En UP se tiene uno mínimo por cada iteración donde se obtienen artefactos ligados a los objetivos de las mismas.

Scheduling (calendarización del trabajo)

Consiste en la organización temporal y asignación de recursos a las actividades del proyecto. Para ello:

1.Determinar actividades a realizar —> 2. Asignar tiempos estimados —> 3. Asignar recursos —> Organizar temporalmente las actividades (gráficos)

Se utilizan diagramas para ilustrar la agenda del proyecto.

Diagramas de actividades

Muestran las actividades y sus dependencias, ademas del camino critico.

GRAFOS PERT: Son utilizados para esto. Es una tarea difícil saber cuales son las actividades a realizar en el desarrollo del software, y todavía mas lo es estimarlas, así como sus dependencias. Con estos diagramas tenemos las listas de precedencias de actividades, así como los tiempos estimados y los recursos asignados. Podemos calcular el tiempo mas temprano y el mas tardío para entregar el proyecto. También se calculan las **holguras: Hij = tj*- ti - tj** Y con ello calculamos el camino critico con holgura total nula. La fecha de inicio mas temprana seria **Aij = ti**, y la mas tardía **Aij* = tj* -tij.**

Diagramas de barras

Muestran la agenda del proyecto teniendo en el eje de ordenadas (Y) las actividades y en el de abscisas (X) el tiempo. Se puede ver la evolución del proyecto, las holguras y os recursos asignados a los mismos.

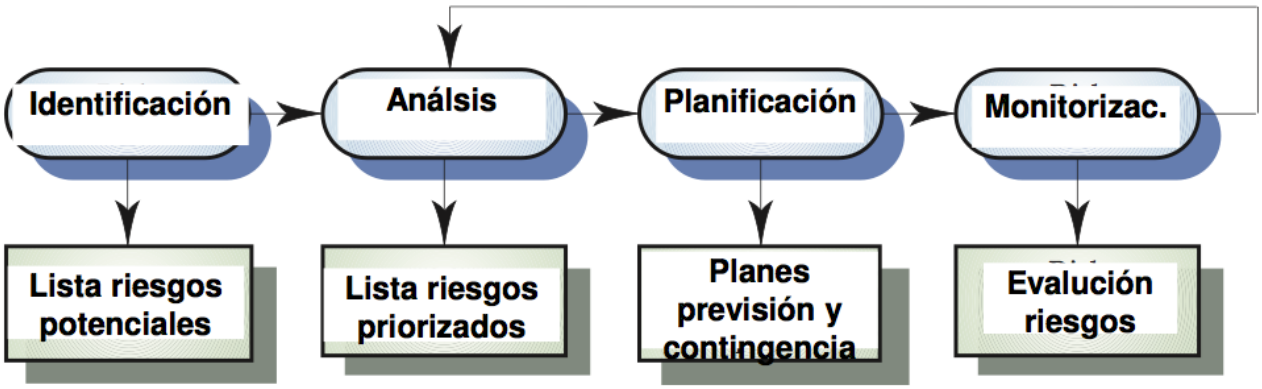
Problemas del Scheduling

Es difícil estimar la dificultad de las actividades. La productividad no es proporcional a los recursos asignados y de hecho puede retrasar más el proyecto si asignamos más personas. ¡Todo puede salir mal!

Gestión de riesgos

Actividad que concierne la identificaciones los riesgo y el desarrollo de planes para minimizar sus efectos dañinos sobre el proyecto (agenda o recursos), el producto (calidad o su realización) y/o al negocio (organización).

tiene 4 fases



tipos de riesgos

1. **Tecnológico:** Tecnologias HW o SW usadas en el desarrollo.
2. **Personal:** Miembros del equipo de desarrollo.
3. **Organización:** Entorno organizacional, la empresa…
4. **Herramientas:** Herramientas CASE u otras en uso.
5. **Requerimientos:** Cambios por parte del cliente.
6. **Estimación:** Estimaciones efectuadas.

Los riesgos se clasifican según:
Probabilidades: muy baja < baja < moderada < alta < muy alta
Efecto: insignificante < tolerable < serio < catastrofico

En la monitorización se intenta evaluar los riesgos para ver si siguen siendo igual de probables o acusan los mismos efectos a medida que el proyecto avanza y se tiene nueva información.
Es un paso importante para poder mantener el dia los planes. Planes que pueden seguir estrategias para evitar el riesgo, reducir sus efectos o reducir el impacto en el proyecto.

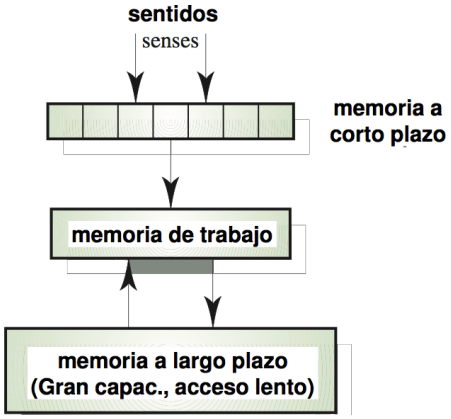
El gestor se encarga principalmente de planificar, estimar y confeccionar agendas, siendo las primeras do actividades que se realizan de forma continuada durante todo el desarrollo del proyecto.

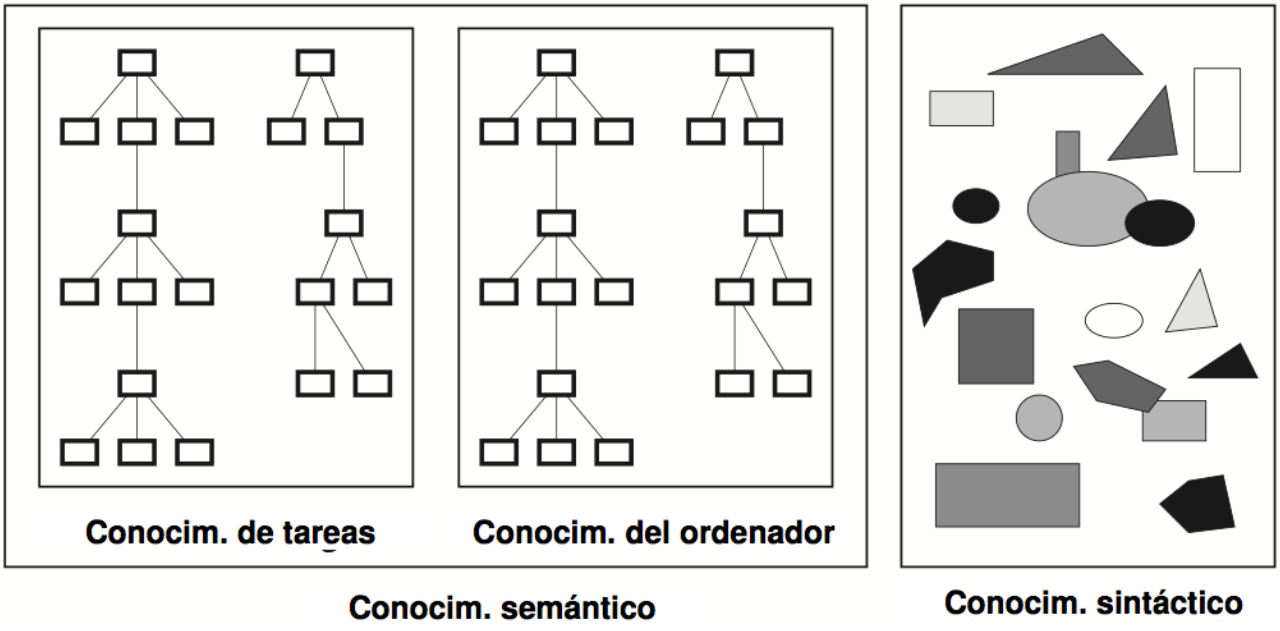
Tema 4. Gestión de recursos humanos

Las personas son la materia prima del desarrollo software, y las tareas del gestor están orientadas a ellas: Solución de problemas, saber motivalas, planificar (aqui tienes que hacer …), estimar (velocidad de trabajo), controlarlas, organizar. Ademas el desarrollo se ve limitado por el conocimiento de las personas pues es una actividad cognitiva.

Pensamiento humano

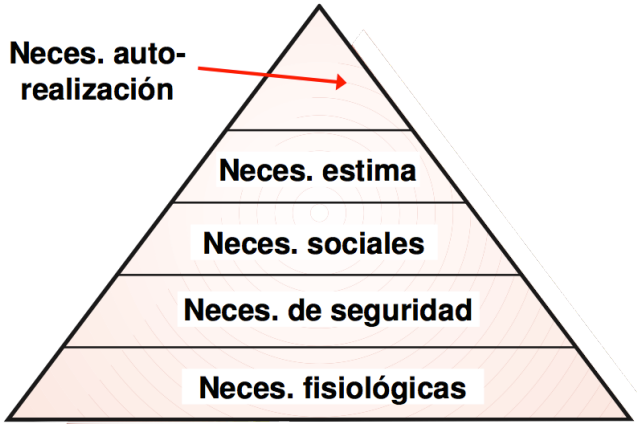
- **Organización de la memoria:** Semántico, obtenido por experiencia y aprendizaje. Sintáctico, por memorizar. El conocimiento sintáctico puede interferir con otro ya existente.
- **Resolución de problemas:** Requiere la integración de conocimiento y experiencias para generar soluciones a nuevos problemas, obteniendo así nueva información. Se genera un modelo semántico de la solución.





Motivación

Es una tarea muy importante y compleja de los gestores. La motivación de forma individual se basa en satisfacer ciertas necesidades que según Maslow son:



fisiológicas: Como dormir, comer, orinar
Seguridad: Estabilidad en el trabajo
Sociales: Sentirse integrado en el grupo
Estima: Ser reconocido en el grupo
Autorrealización: Gustarse, sentirse completado, disfrutar con tu trabajo

La jerarquía de necesidades es una simplificación de la realidad, a parte de esto tenemos que tener en cuenta la personalidad. Encontramos 3 tipos de personalidades principales:

1. **Orientadas a la tarea:** Disfrutan resolviendo problemas, ingenieros.
2. **Orientadas a si mismo:** Gustan de si mismos, abogados.
3. **Orientadas a la autorrealización:** Gustan de trabajar con otros, psicólogos.

No solo afectan estos factores, también dependerá de circunstancias externas como son la cultura o eventos externos, así como los derivados por formar parte de un grupo.

El trabajo en grupo

El desarrollo software es un trabajo colectivo, no es posible que una persona sola complete un trabajo al 100% en solitario. Así pues son claves la interacción de los miembros del grupo, como su composición. Existen estudios que indican que el 20% del tiempo los trabajadores realizan actividades no productivas, el 30% trabajan solos, y el 50% restante lo pasan interactuando con los demás. Así pues, como vemos, es un factor vital.

Existen otros 5 factores vitales, que también influyen en el trabajo en grupo:

1. **Composición del grupo:** No es bueno que todos tenga la misma motivación (o personalidad) pues los orientados a a la tarea querrán hacer las cosas a su manera, los orientados a si mismos pelearan por el liderazgo, y los orientados a la interacción se las pasarán charlando. **¡EQUILIBRIO!** Aún así los gestores deben aprender a trabajar con lo que tienen pues no siempre es posible conseguir y emplear los recursos que se desean.
2. **Líder del grupo:** Debe ser establecido por respeto y no basarse en un titulo que proporcione **status**. Si aparece de forma democrática funcionará mejor que si es impuesto. Para proyectos software es recomendable tener un **líder técnico** y uno **administrativo** y seria ideal que pudiera desarrollar ambas trayectorias en paralelo.
3. **Cohesión del grupo:** Un grupo cohesionado considera mas importante el grupo que el individuo . Asi se pueden crear **estandares de calidad** que todos comprendan y acepten, **aprenden unos de otros** y se reduce la inhibición, y ademas se puede practicar la "**programación sin ego**", por el bien del proyecto. La **sinceridad** es un aspecto clave para conseguirlo, así como el **desarrollo de una identidad de grupo** con evento y actividades que unan al grupo
4. **Comunicación del grupo:** 50% del tiempo de trabajo se pasa interactuando así que una buena comunicación **es muy importante.** Además, esto ayudará a conseguir una cohesión del grupo y promueve el entendimiento. Influyen los status, personalidades, sexos y canales de comunicación.
5. **Organización del equipo:** Normalmente no mas de 8-10 personas para reducir las necesidades de comunicación que conllevan tanto tiempo. Así pues, se suelen dividir proyectos grandes en subgrupos en los que cada subgrupo trabaja. El líder del grupo será un enlace con el resto, y dentro del grupo se trabaja la forma democrática e informal. El trabajo se reparte por los propios integrantes. **Funciona mejor si son experimentados.** Dentro de la XP, encontramos programación por parejas, responsabilidad colectiva, decisiones de gestión…

Selección de personal

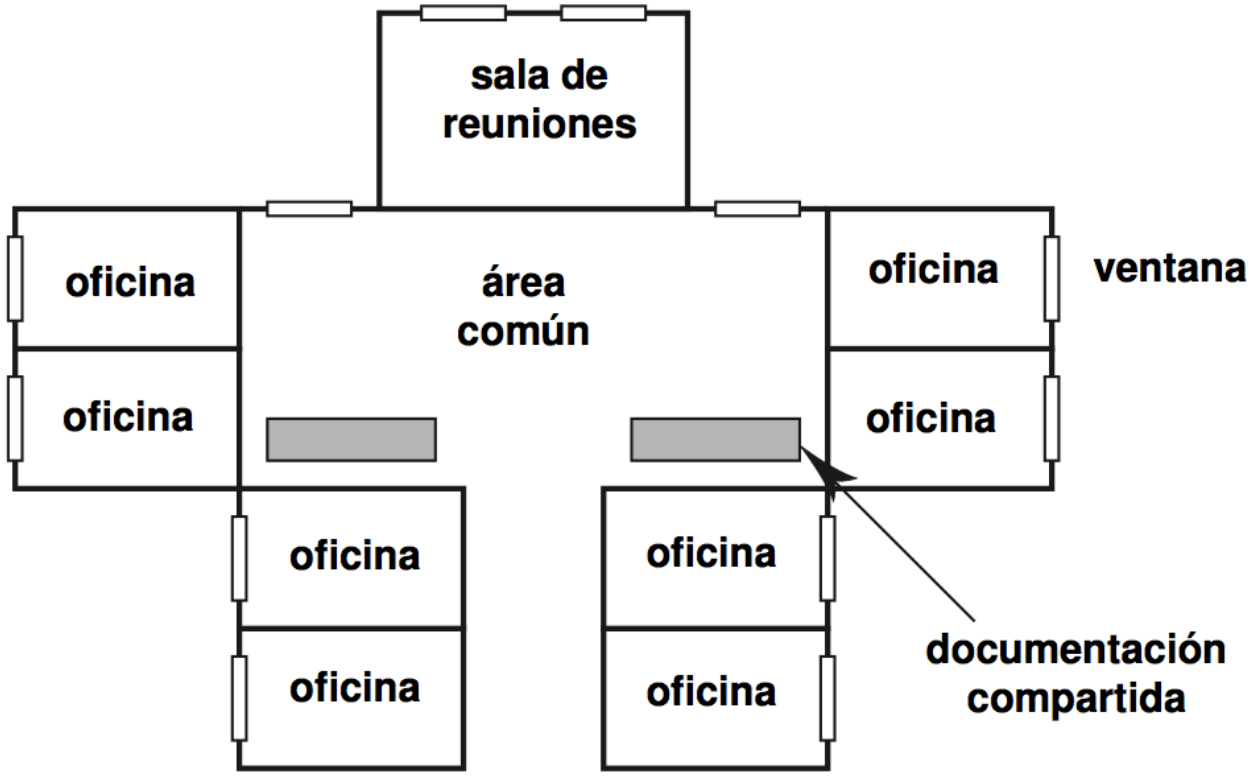
Es una responsabilidad importante del gestor. Sus decisiones se basarán en la información proporcionada por el candidato en su **CV**, se obtendrá en una **entrevista** y **recomendaciones de otra gente.** Algunos usan test psicológicos y/o de aptitud pero su efectividad no está probada.
Factores que debemos tener también en cuenta:

- Experiencia en el dominio de la aplicación
- Experiencia en la plataforma
- Experiencia en el lenguaje de programación
- Estudios
- Capacidad de comunicación
- Adaptabilidad
- Actitud
- Personalidad

Entorno de trabajo

Esta comprobado que el **entorno físico** es importante para lograr mayor productividad y satisfacción. Se ha de considerar los siguientes aspectos relacionados con el individuo y el lugar de trabajo:

- Confort
- Privacidad
- Facilidades
- Iluminación
- Climatización
- Mobiliario



Tema 5. Gestión de la configuración del software

Consiste en una actividad de autoprotección de los artefactos para **identificar** cambios, **controlarlos**, **garantizar** que se implementará adecuadamente e **informar** de ello a los interesados. ¡No tiene nada que ver con tareas de mantenimiento!.

Planificación

La planificación de la GC se organiza en:
1. Definición de **lo que se va a gestionar** (Elementos de configuración(EC)) y esquema forma de identificación
2. Identificación de **los responsables** de los procedimientos de la GC.
3. Descripción de **como mantener los registros** de documentos.
4. Descripción de las **herramientas usadas** para la GC.
5. Definición de la base de datos de las configuraciones (**BDC**).

La identificación de los EC suele llevar un **nombrado jerárquico** que identifique cada uno de forma inequívoca, así pues suelen ir asociados a proyectos particulares, **desfavoreciendo la reusabilidad**. Todo documento o artefacto que se genere es susceptible de ser gestionado pero no todos son necesarios. Típicamente se gestionan:
- Planes del proyecto
- Especificaciones
- Diseños
- Programas o ejecutables
- Datos o conjuntos de pruebas
- Manuales

La BDC se utiliza para registrar toda la **información relevante** que esté **relacionada con las configuraciones**. Su creación puede darse como un **sistema aparte o integrarlo** con la **gestión de versiones** y el **sistema de control** que almacena los documentos formales del proyecto.

Control de versiones

Implica la identificación y el seguimiento de las diferentes versiones y releases del sistema. De esta forma es posible recuperar alguna versión anterior, por ejemplo:
Nueva versión —> nuevo fuente + Construcción del sistema (1)
Nueva release —> (1) + ficheros y datos de configuración + nueva documentación

Las **versiones** son instancias del sistema que difieren de otras de alguna manera como puede ser **funcionalidades,** nuevas mejoras de **rendimiento** o **reparaciones de fallos.**

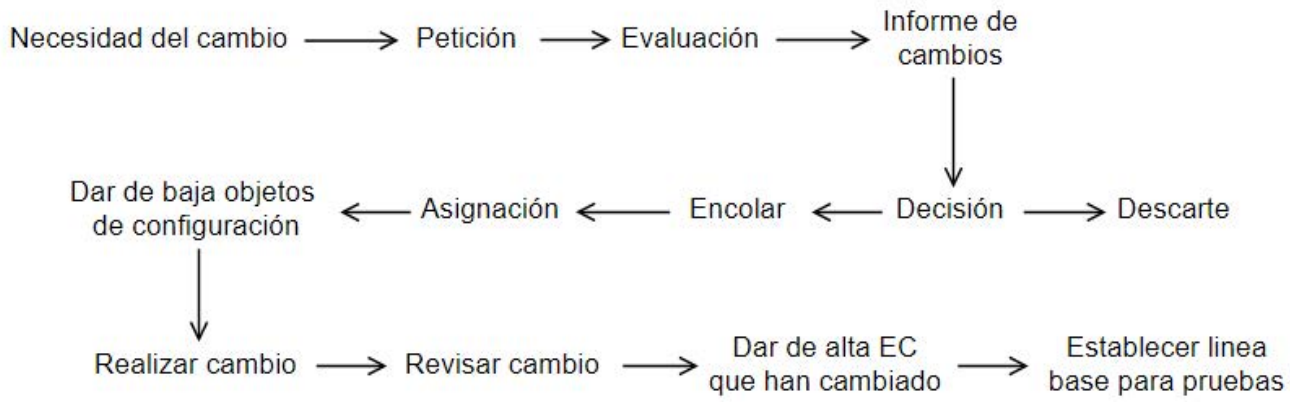
La **entrega,** además se distribuye al cliente junto con nueva documentación y ficheros pues no es una versión que se quede para la gestión y control interno del desarrollo. El grupo **responsable de la GC** se encarga de **decidir** cuando es necesaria una nueva **versión o release**.

Como herramientas tenemos **RCS** (Revision Control Source):
- Identificación de versiones y releases
- Cambios controlados
- Gestión de almacenamiento
- Registro de la historia de cambios
- Grabación del código fuente en su versión mas reciente
- Soporte de desarrollo paralelo de diferentes releases
- Capacidad de mezcla de versiones

Existen otras herramientas en Unix como son SCCS, pero la mas conocida y usada es RCS. Ambas permiten trabajar con texto ASCII.

Control de cambios

Los cambios son algo natural en la vida de un sistema software. Cuando un sistema ya está en marcha y es operativo, realizar cambios en el mismo se convierte en una tarea mas costosa. Se han de crear mecanismos para controlar dichos cambios, que ademas se aseguren de que se hacen de forma adecuada y que realmente merecen la pena, llevando un registro de los mismos junto a un estudio y un análisis de los costes y beneficios.



Construcción del sistema

Proceso de **compilar y linkar** o vincular los componentes del software que se ejecutara sobre una configuración destino particular. Al construir se ha de considerar si:

1. Se han incluido todos los ficheros.
2. Estos están en la versión adecuada.
3. Tenemos disponibles todos los ficheros de datos.
4. Los datos se nombran igual en el componente y la máquina destino.
5. La versión del compilador es la adecuada.

Normalmente se emplea MAKE para ello, indicando las dependencias entre los componentes, escribiendo los ficheros “makefile”. De esta forma se **fuerza la recompilación** de los ficheros cuyo código ha cambiado tras la creación de su objeto. Estos ficheros se pueden hacer **muy complejos,** y ademas MAKE simplemente se basa en comprobar que la fecha de modificación del fichero fuente y del objeto creado son diferentes. A veces no es necesario recompilar (podemos haber añadido comentarios).

Beneficios de los GCS

- Reducen el esfuerzo de gestionar y realizar los cambios consiguiendo un aumento de la productividad.
- Una mejora de la integridad y seguridad del software ya que todos los cambios no son aceptados, lo que da lugar a un incremento de la calidad.
- Todo el proceso se mantiene en un registro lleno de información, permitiendo llevar una mejor gestión de control
- Al mantener una BDC, se tiene un mejor registro y seguimiento de los documentos y artefactos de forma que en todo momento se suben las versiones y entregas disponibles junto a su localización entre otra información detallada del mismo.

Tema 6. Monitorización y control

El 70% de los proyectos cuestan mas de lo presupuestado y se entregan mas tarde de lo planificado. Además el 52% cuestan un 189% mas de lo presupuestado. Otros ni siquiera se entregan.

La mayor parte de los problemas de estos proyectos se deben a un **mal seguimiento, monitorización y control del mismo.**
Cuando monitorizamos hacemos un seguimiento de si la **agenda real** (con datos reales) se **ajusta** a la **agenda planificada** (con nuestra información planificada). Para ello se emplean métricas - mediciones que aportan valor cuantitativo sobre un atributo de un sistema/componente/proceso:

- **Fechas** de inicio y fin
- **Holguras** totales y libres
- **Duraciones** de las tareas
- Análisis de valor acumulado **(EVA)**

Estas medidas serán empleada con el objetivo de tomar las **decisiones adecuadas** que nos lleven a cumplir los plazos temporales, costes y expectativas del usuario que teníamos previstos. Una vez tengamos todos estos datos, se confeccionan la agenda provista y se programa su inicio.

Earned Value Analisis (EVA)

Es una métrica que proporciona información cuantitativa del progreso del proyecto. Nos servirá para ir detectando dificultades en la agenda antes de que se conviertan en grandes problemas de forma que el gestor pueda tomar acciones cuanto antes.
Sin este estudio, realmente no sabremos nada sobre lo que pasa en el proyecto. Tipos de información:

1. **BCWS** (Budgeted Cost of Work Scheduled): Indica cuanto trabajo del planificado se debería haber completado en términos de valor. VALOR PLANIFICADO
2. **BCWP** (Budgeted Cost of Work Performed): Indica el valor del trabajo que realmente se ha llevado acabo. VALOR ACUMULADO
3. **ACWP** (Actual Cost of Work Performed): Indica cuanto se ha gastado hasta el momento. COSTE REAL

Estos análisis se toman siempre teniendo un instante de tiempo concreto como referencia.

Tenemos indicadores de **PROGRESO:**

- **SV:** Schedule Variance. BCWP - BCWS
- **SPI:** Schedule Performance Index. BCWP/BCWS

Si el SPI < 1, vamos retrasados. Si en cambio el SPI > 1, vamos adelantados.

Tenemos indicadores de **PRODUCTIVIDAD:**

- **CV:** Cost Variance. BCWP - ACWP
- **CPI:** Cost Performance Index. BCWP/ACWP

Si el CPI > 1, estamos gastando menos. Si en cambio el CPI < 1, gastamos más.

Si tenemos una **buena productividad y un progreso lento:** ¡Nos falta gente!

Existe otro termino que es el BAC (Budget At Completion) que indica el valor del trabajo planificado al terminar el proyecto. Si para un instante y una actividad, el BAC y el BCWS son iguales significa que la tarea debería haberse completado. Si BAC > BCWS es que aun no se debe haber terminado, y si BCWS != 0 es que aún no se ha empezado.

Control de agendas

El control de la agenda conlleva realizar las acciones necesarias para reducir y evitar las discrepancias entre la agenda real y la planificada. Las actividades críticas no se deben retrasar. Y si es necesario podemos resignar recursos y dinero con el fin de evitar esto desde alguna actividad que vaya con mejores valores EVA, así como holguras de forma que su retraso no afecte a otras actividades o al menos no al proyecto.