



E.T.S. INGENIERÍA  
INFORMÁTICA  
UNIVERSIDAD DE MÁLAGA

# PROJECT MANAGEMENT

GRADO EN INGENIERÍA DEL SOFTWARE

## TEMA 1. INTRODUCCIÓN A LA GESTIÓN DE PROYECTOS SOFTWARE

*human  
resources*

*scope*

*quality*

*cost*

*time*

*risk*

*procurement*

*communication*



# CONTENIDO

- Introducción a la gestión de proyectos software
  - Definición de la gestión de proyectos software
  - Importancia de la gestión de proyectos de software
  - Cuestiones en las prácticas de gestión de proyectos
  - Etapas en el proyecto de software:
    - Solicitud de propuesta (RFP)
    - Definición del proyecto
    - Requisitos
      - Trazabilidad
      - Priorización (MoSCoW, WSJF)
    - Estudio de viabilidad
    - Arquitectura y decisiones técnicas
    - Planificación
    - Ejecución del proyecto
    - Cierre del proyecto
  - Modelos de ciclos de vida de proyectos
  - Partes interesadas de un proyecto
  - Deuda técnica



# OBJETIVOS DEL TEMA

- Comprender qué implica gestionar un proyecto software
- Identificar las etapas y principales decisiones
- Relacionar requisitos, alcance, viabilidad, arquitectura y planificación
- Reconocer el papel de stakeholders y deuda técnica
- Comparar modelos de ciclo de vida



# ¿QUÉ ES UN PROYECTO?

- Esfuerzo temporal para crear un producto, servicio o resultado único
- Tiene objetivos, restricciones, recursos y partes interesadas
- No es lo mismo que una operación recurrente o repetitiva



# PROYECTOS SOFTWARE: PARTICULARIDADES

- El producto es principalmente intangible
- Requisitos y solución evolucionan con frecuencia
- Alta incertidumbre técnica y organizativa
- El progreso no siempre es visible
- El coste del cambio suele crecer con el tiempo



# GESTIÓN DE PROYECTOS

- Aplicación de conocimientos, habilidades, herramientas y técnicas para cumplir los requisitos del proyecto
- Equilibrando/cumpliendo restricciones de alcance, tiempo, coste, calidad, riesgos y recursos



# GESTIÓN ≠ PROGRAMACIÓN

- Programar es una actividad del proyecto
- Gestionar implica tomar decisiones sobre:
  - qué hacer
  - por qué
  - cuándo
  - con qué recursos
  - quién participa
  - qué riesgos aceptar
- Gestionar implica reaccionar para controlar:
  - Plazos, costes, etc.
  - Personas / interacciones
  - Imprevistos / desviaciones
  - Cambios



# LA GESTIÓN COMO SISTEMA DE DECISIONES

- Proceso
  - Objetivos → requisitos → solución → plan → ejecución → resultados
- Cada decisión condiciona las siguientes
- La gestión debe mantener coherencia entre todas ellas



# RESTRICCIONES DEL PROYECTO

- Alcance
- Plazo
- Coste
- Calidad
- Recursos
- Riesgos



# LA «TRIADA DE RESTRICCIONES» Y SU EVOLUCIÓN

- Modelo clásico: alcance · tiempo · coste
  - Pero la calidad y el riesgo no pueden tratarse como variables secundarias
- En proyectos modernos: múltiples restricciones y objetivos en tensión



# DEFINIENDO EL ÉXITO DEL PROYECTO

- ¿Terminar a tiempo y dentro del presupuesto = éxito?
- También importan:
  - Valor para el negocio/usuario
  - Calidad y sostenibilidad
    - Múltiples aspectos de calidad (seguridad, mantenibilidad, etc.)
    - Deuda técnica
  - Satisfacción de stakeholders
  - Cumplimiento legal y organizativo



# ¿POR QUÉ FALLAN LOS PROYECTOS SOFTWARE?

- Objetivos ambiguos
- Requisitos inestables o mal entendidos
- Estimaciones demasiado optimistas
- Riesgos ignorados o mal gestionados
- Problemas técnicos subestimados
- Comunicación deficiente
- Problemas del personal



# EL COSTE DE LOS ERRORES

- Un error temprano puede ser barato de corregir
- El mismo error puede ser muy costoso durante integración o producción
- Por eso la gestión busca reducir incertidumbre cuanto antes



# CUESTIONES CLAVE EN LA PRÁCTICA

- ¿Qué problema resolvemos?
- ¿Para quién(es)?
- ¿Qué se incluye y qué queda fuera?
- ¿Cómo sabremos que funciona?
- ¿Es viable?
- ¿Qué puede salir mal?
  - ¿Cómo evitamos o mitigamos lo que pueda salir mal?
- ¿Qué necesitamos para ejecutarlo?



# PROYECTO: DE LA OPORTUNIDAD AL PRODUCTO

- Idea / necesidad / oportunidad
- ↓
- Solicitud de propuesta (RFP)
- ↓
- Definición y viabilidad
- ↓
- Planificación y ejecución
- ↓
- Entrega y cierre



# 1. RFP - REQUEST FOR PROPOSALS

- Documento mediante el que una organización solicita propuestas
- Suele describir necesidad, objetivos, alcance esperado y condiciones
- Permite comparar alternativas de proveedores o soluciones



# CONTENIDO HABITUAL DE UNA RFP

- Contexto y problema
- Objetivos y alcance esperado
- Requisitos
- Entregables
- Plazos e hitos
- Criterios de evaluación
- Presupuesto / condiciones contractuales



# RESPONDER A UNA RFP

- Entender el problema antes de proponer la solución
- Identificar supuestos (asunciones) y ambigüedades
- Proponer alcance, solución, plan, equipo y costes
- Explicitar riesgos y exclusiones



# DE LA RFP A LA DEFINICIÓN

- La RFP describe una necesidad
- El proyecto debe convertirla en un marco gestionable
  - objetivos verificables
  - alcance
  - requisitos
  - stakeholders
  - criterios de éxito



# PROJECT CHARTER - ACTA DE CONSTITUCIÓN

- Documento que autoriza formalmente el proyecto
- Define propósito, objetivos, alcance inicial, stakeholders y responsabilidades
- Sirve como referencia para decisiones posteriores



# ELEMENTOS DE UNA BUENA DEFINICIÓN

- Problema / oportunidad
- Objetivos SMART cuando sea apropiado
  - Específicos (**S**pecific)
  - Medibles (**M**easurable)
  - Alcanzables (**A**chievable)
  - Realistas (**R**ealistic)
  - De duración limitada (**T**ime-bound).
- Alcance y exclusiones
- Entregables
- Supuestos y restricciones
- Criterios de éxito



# ALCANCE: QUÉ SÍ Y QUÉ NO

- IN SCOPE
  - Funcionalidades y entregables comprometidos
- OUT OF SCOPE
  - Elementos explícitamente excluidos
- El alcance no definido es una fuente frecuente de conflictos



E.T.S. INGENIERÍA  
INFORMÁTICA  
UNIVERSIDAD DE MÁLAGA

# Requirements



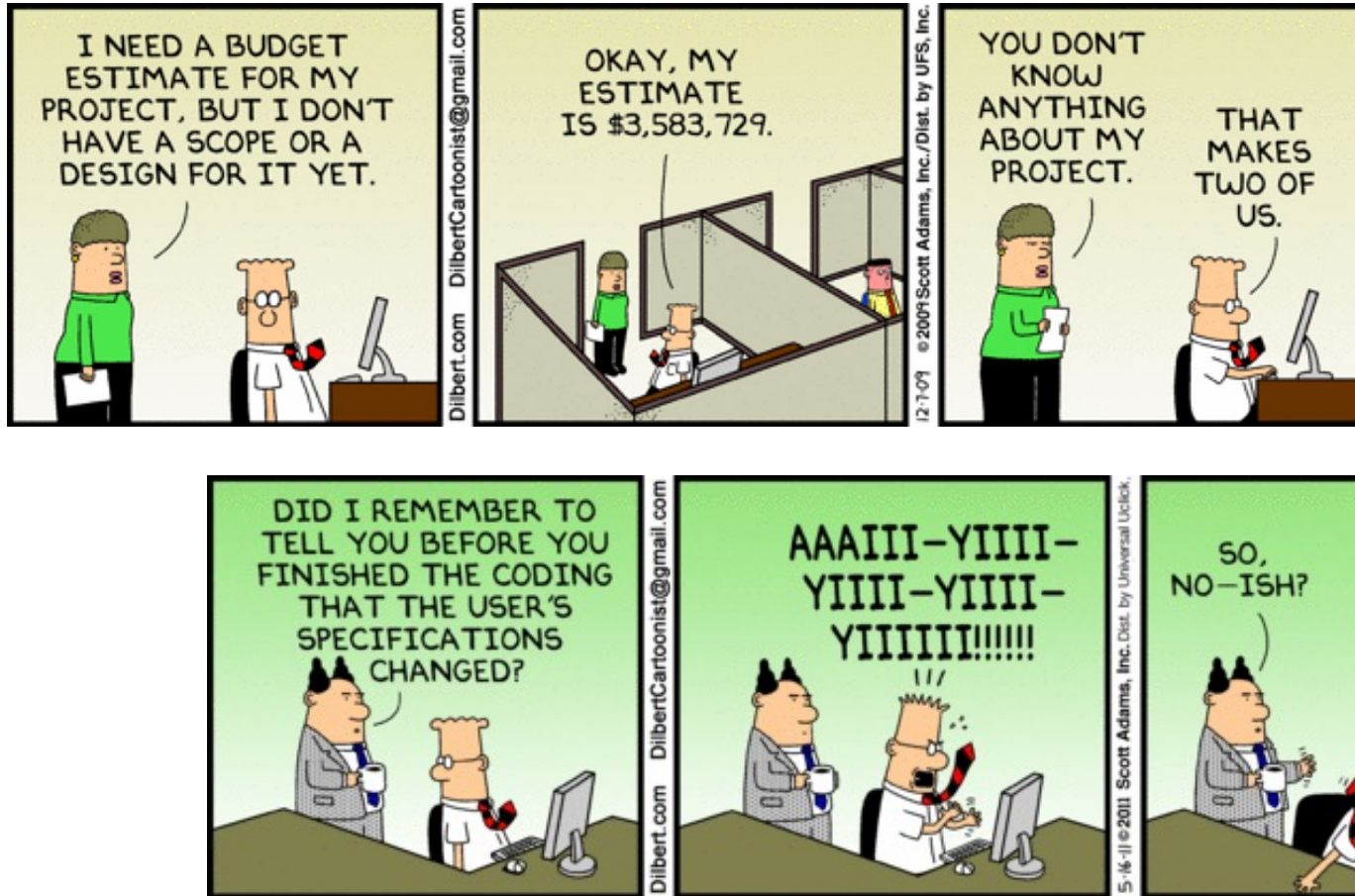


# SITUÉMONOS



© Scott Adams, Inc./Dist. by UFS, Inc.

# SITUÉMONOS





# REQUISITOS

- Describen necesidades, capacidades o condiciones que debe satisfacer el sistema
- Deben ser comprensibles, verificables y trazables
- Tipos habituales: funcionales y no funcionales
- Sus relaciones son tan cruciales como los propios requisitos
- La trazabilidad es esencial



# REQUISITOS FUNCIONALES VS. NO FUNCIONALES

- Los **requisitos funcionales** describen las **capacidades** del sistema  
¿QUÉ HACE?.
- Los **requisitos no funcionales** describen las **cualidades** del sistema  
¿CÓMO HACE LO QUE HACE?
- Ej.: «registrar pagos» vs. «procesar el 95 % de pagos en < 2 s»
- Algunos requisitos tienen una mezcla de ambos tipos
  - Ej. Aspectos de seguridad.
  - Para facilitar su tratamiento los consideraremos como RNF.
- Los RNFs son los que suelen influir más en la **complejidad** del proyecto.
  - Además, es más difícil verificar que se cumplen.



# REQUISITOS FUNCIONALES (RF)

- Describen los servicios y la funcionalidad del sistema
- Dependen del tipo de software, del tipo de sistema donde se usará y de los posibles usuarios
- Los requisitos funcionales de usuario pueden ser declaraciones de alto nivel sobre lo que realizará el sistema ...
  - pero los requisitos funcionales del sistema deben describir los servicios del sistema detalladamente



## EJEMPLOS DE RF

- *El usuario tendrá la posibilidad de buscar en el conjunto de toda la base de datos o en cualquier subconjunto que seleccione de ella.*
- *El sistema proporcionará visores adecuados que permitan leer los documentos almacenados en el repositorio de documentos*
- *A cada pedido se le deberá asignar un identificador único (ID\_PEDIDO) que el usuario podrá copiar al área de almacenamiento permanente de la cuenta*



# REQUISITOS NO-FUNCIONALES (RNF)

- Definen propiedades y restricciones del sistema
  - Fiabilidad, tiempo de respuesta, capacidad de almacenamiento, ...
- Algunos requisitos del proceso pueden obligar a utilizar estándares de calidad o una herramienta CASE particular o un lenguaje de programación
- Los requisitos no-funcionales pueden ser más críticos que los funcionales.
  - Si no se cumplen, invalidan el sistema
  - Su cumplimiento tiene gran influencia en la arquitectura del sistema



# CLASIFICACIÓN DE RNF POR OBJETO

- **Requisitos del producto**
  - Especifican el comportamiento del producto
  - Ej.: Velocidad de ejecución, Tasa aceptable de fallos, ...
- **Requisitos de la organización**
  - Son consecuencia de las políticas de y procedimientos de la propia organización
  - Ej.: estándares de proceso, métodos de diseño, lenguajes de programación, tipo de documentación
- **Requisitos externos**
  - Se derivan de factores externos al producto y a su proceso de desarrollo
  - Ej.: Requisitos legales (LOPD), éticos, ...

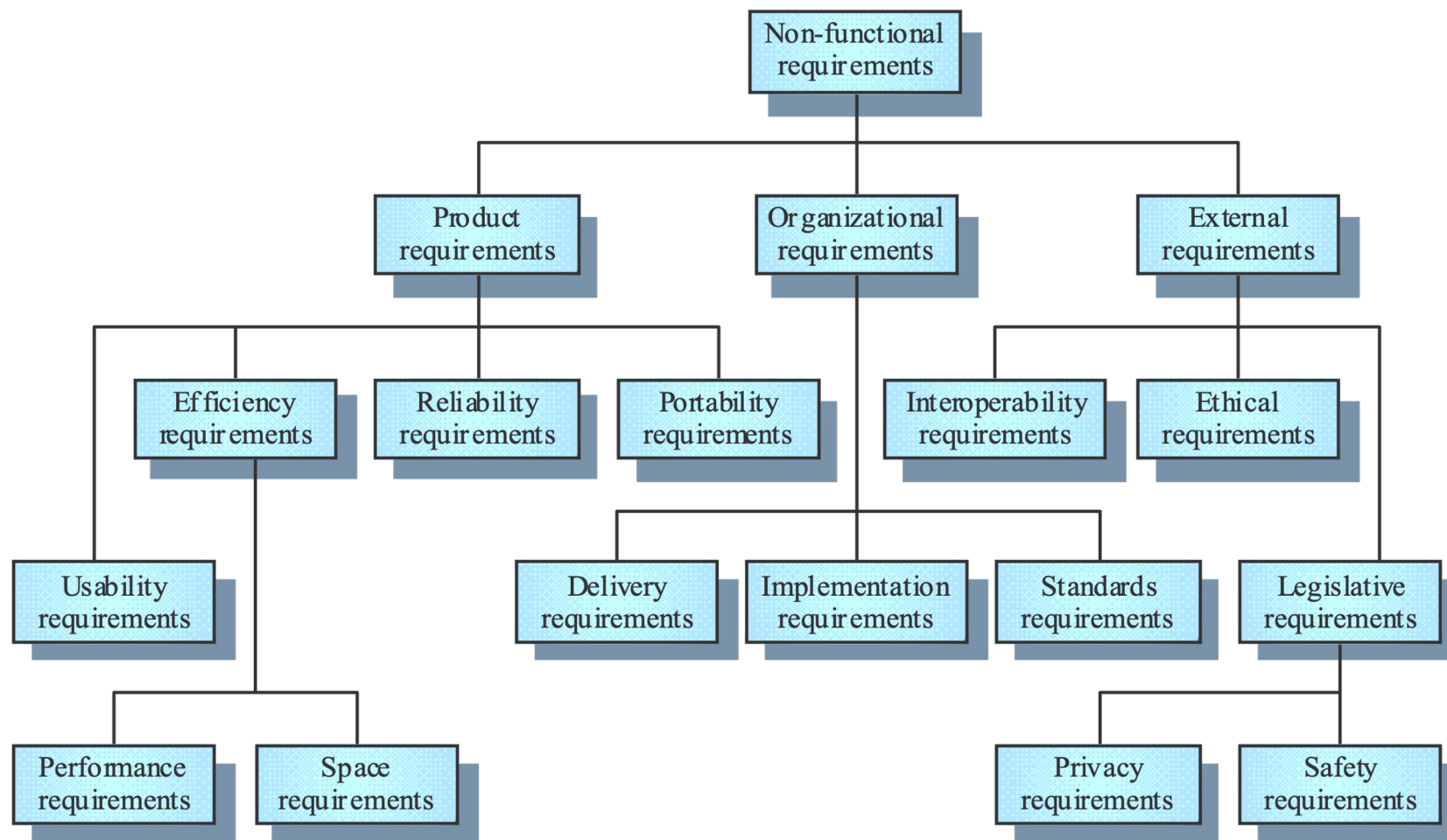


# EJEMPLOS

- Requisito del producto
  - 4.C.8. Será necesario que la comunicación entre el APSE y el usuario se exprese utilizando el conjunto estándar de caracteres UTF-16BE
- Requisito de la organización
  - 9.3.2 El proceso de desarrollo y los documentos entregados serán conformes con el estándar ISO-XXXX
- Requisito externo
  - 7.6.5 El sistema no revelará datos de carácter personal de los clientes a los usuarios con nivel 3 o inferior excepto el nombre y número de referencia



# JERARQUÍA DE RNF POR TIPO





# REQUISITOS Y CRITERIOS DE ACEPTACIÓN

- Es fácil comprender que se deben evitar requisitos vagos: «rápido», «seguro», «fácil de usar»
- Requisito → condición verificable → necesita **criterios de aceptación**
- Los **criterios de aceptación** definen realmente el requisito porque definen cuándo se considera satisfecho
  - TDD aplica las pruebas como **criterios de aceptación**



# TRAZABILIDAD

- Permite seguir la relación entre elementos del proyecto
- Necesidad → objetivo → requisito → diseño → implementación → prueba
- Ayuda a detectar requisitos sin cobertura y el impacto de los cambios



# MATRIZ DE TRAZABILIDAD

- Ejemplo:
- OBJ-02 → REQ-07 → MOD-03 → TEST-21
- Preguntas:
  - ¿Todos los requisitos tienen origen?
  - ¿ Todos los requisitos tienen verifican?
  - ¿Qué pruebas se ven afectadas por un cambio?



# PRIORIZACIÓN DE REQUISITOS

- No todo puede hacerse primero
- La prioridad debe reflejar valor, urgencia, riesgo y coste
  - Se pueden aplicar diferentes estrategias para priorizar los requisitos:
    - Primero los de más valor
    - Primero los de más riesgo
    - ...
- La priorización debe ser explícita y revisable



# ESTRATEGIAS

- No existe una única estrategia para priorizar.
  - **Valor** → lo que aporta más valor primero
  - **Riesgo** → lo más arriesgado primero
  - **Coste de retraso** → lo que más cuesta posponer
  - **Dependencias** → lo que desbloquea otras funcionalidades
  - **Esfuerzo** → resultados rápidos / *quick wins*
  - **Aprendizaje** → reducir incertidumbre primero



# EJEMPLOS DE TÉCNICAS

- **MoSCoW** → clasificación por importancia
- **WSJF** → Cost of Delay / Job Size
- **Value vs. Effort** → valor frente a esfuerzo
- **Risk-based prioritization** → probabilidad × impacto
- **Kano** → satisfacción del cliente
- **Cost of Delay** → impacto económico/temporal de retrasar
- **Opportunity scoring** → importancia para el cliente vs. satisfacción actual



# MOSCOW

- No proporciona una estrategia sino una semántica estándar para representar la prioridad de forma explícita-
- Clasifica la prioridad en:
  - MUST · imprescindible
  - SHOULD · importante, no imprescindible
  - COULD · deseable
  - WON'T · no se hará ahora
- Útil para negociar alcance y gestionar expectativas



# ESTUDIO DE VIABILIDAD

- Pregunta fundamental: ¿debemos hacer este proyecto?
  - Viabilidad técnica
  - económica
  - temporal
  - organizativa
  - legal / regulatoria



# VIABILIDAD TÉCNICA

- ¿Podemos construirlo con la tecnología disponible?
- ¿Tenemos conocimientos y recursos?
- ¿Existen integraciones, datos o restricciones críticas?
- ¿Qué incertidumbres requieren un prototipo o prueba de concepto?



# VIABILIDAD ECONÓMICA Y TEMPORAL

- Beneficios esperados vs. costes
- Costes:
  - de desarrollo + operación + mantenimiento
- ¿Hay una fecha objetivo?
  - Oportunidad de negocio
    - Competencia
    - Circunstancias objetivo
  - Límites externos
  - Límites internos
- ¿Es posible cumplir la fecha objetivo?
  - Una fecha fija puede obligar a reducir alcance o aumentar recursos



# ARQUITECTURA SOFTWARE

*“As the size and complexity of software systems increases, the design problem goes beyond the algorithms and the data structures of the computation: designing and specifying the overall system structure emerges as a new kind of problem... This is the software architecture level of design.”*

Garland y Shaw, 1994



# CONCEPTO DE ARQUITECTURA SOFTWARE

*“Estructura de los componentes de un programa o sistema, sus interrelaciones, y los principios y reglas que gobiernan su diseño y evolución en el tiempo.”*

Garlan y Perry, 1995

*“Estructura o estructuras de un sistema, lo que incluye sus componentes de software, las propiedades observables de dichos componentes y las relaciones entre ellos. “*

Bass, Clements y Kazman, 1998

*“Para mí el término arquitectura conlleva la noción de los elementos que forman el núcleo del sistema, las piezas que son difíciles de cambiar. Los cimientos sobre los que se edifica el resto.”*

Martin Fowler, 2001



# ARQUITECTURA Y DECISIONES TÉCNICAS

- La arquitectura define la estructura fundamental del sistema
- Las decisiones técnicas deben responder a requisitos y restricciones
- No basta con elegir tecnologías “de moda”
  - Los clientes tienden a pedir las (“buzzwords”)



# ARCHITECTURE DECISION RECORDS · ADR

- Un registro de decisión de arquitectura (ADR) es un documento que recoge una decisión importante tomada junto con su contexto y sus consecuencias.
- Registra y traza decisiones importantes
  - Contexto → decisión → alternativas → consecuencias
- Facilita la revisión, mantenimiento y aprendizaje del equipo de desarrollo.

<https://github.com/architecture-decision-record/architecture-decision-record#how-to-start-using-adrs>



# MODELOS DE CICLO DE VIDA

- Predictivo / secuencial / cascada
- Iterativo
- Incremental
- Ágil
- Híbrido



# MODELOS DE CICLO DE VIDA

- La elección del modelo de ciclo de vida no es una decisión trivial ni única, sino que debe adaptarse al contexto del proyecto considerando factores como:
  - la estabilidad de los requisitos,
  - el tamaño del equipo,
  - el riesgo tecnológico y
  - el plazo disponible,
- ya que un modelo inadecuado (ya sea predictivo, ágil o híbrido) puede condenar al proyecto al fracaso antes de escribir la primera línea de código, mientras que el acertado actúa como un habilitador natural del éxito.



# PREDICTIVO VS. ADAPTATIVO

- **PREDICTIVO:**
  - se intenta definir y planificar gran parte del trabajo al inicio
- **ADAPTATIVO:**
  - se aprende y ajusta progresivamente

¿Podéis darme ejemplos de proyectos en los que sería mejor uno que otro?



# STAKEHOLDERS · PARTES INTERESADAS

- Entidades (personas, grupos u organizaciones) que pueden afectar o verse afectados por el proyecto.
- Ejemplos: cliente, usuarios, dirección, equipo, proveedores, reguladores (gobiernos, asociaciones, etc.)
- Las partes interesadas tienen objetivos que pueden ser opuestos e incompatibles
  - En ingeniería de seguridad se consideran también las partes interesadas maliciosas además de las legítimas



# STAKEHOLDERS Y PUNTOS DE VISTA (VIEWPOINTS)

- Dos entidades pueden ver el mismo proyecto de forma diferente
  - Identificar perspectivas ayuda a descubrir objetivos y conflictos
  - El análisis de stakeholders debe influir en requisitos y decisiones



# FASE DE DEFINICIÓN DEL SISTEMA

- La fase de definición del sistema es el punto de partida fundamental donde se identifican todos los stakeholders y se explicitan sus objetivos, expectativas y restricciones.
- El objetivo es analizar de forma crítica posibles conflictos, prioridades incompatibles o requisiciones contradictorias, resolviéndolos antes de que el desarrollo comience, ya que es mucho más económico y eficiente alinear intereses en el papel que corregir desviaciones cuando el código ya está escrito.



# ANALIZAR STAKEHOLDERS

- Interés: ¿qué esperan o necesitan?
- Influencia: ¿pueden afectar decisiones?
- Impacto: ¿cómo les afecta el resultado?
- Estrategia: informar, involucrar, consultar, gestionar



# DEUDA TÉCNICA

- Coste futuro provocado por decisiones técnicas que priorizan una solución inmediata
- Puede ser deliberada o accidental
- No toda deuda es mala: debe gestionarse conscientemente



# DEUDA TÉCNICA

- La deuda técnica es el coste acumulado por tomar atajos o decisiones subóptimas en el desarrollo.
  - Esto ralentiza la productividad futura, aumenta la fragilidad del sistema y consume recursos que podrían destinarse a nuevas funcionalidades, por lo que gestionarla activamente es tan crítico como entregar valor en cada iteración.
- La deuda técnica es como comprar a crédito:
  - te permite avanzar rápido hoy, pero los intereses (en forma de bugs, complejidad y pérdida de velocidad) se pagan en el futuro, y si no la amortizas periódicamente, acabas hipotecando todo el proyecto.



# TIPOS/FUENTES DE DEUDA TÉCNICA

- Código / diseño
- Arquitectura
- Pruebas
- Infraestructura / dependencias
- Documentación
- Procesos y conocimiento



# GESTIONAR LA DEUDA TÉCNICA

- Registrar la deuda
- Estimar impacto y riesgo
- Priorizar su tratamiento
- Planificar su reducción
- Evitar que se convierta en deuda “invisible”



# RELACIÓN ENTRE DEUDA Y GESTIÓN

- Presión de plazo → decisiones rápidas
- Decisiones rápidas → deuda potencial
- Deuda acumulada → menor velocidad y mayor riesgo
- Gestión responsable → hacer explícito el compromiso (trade-off)



# PLANIFICACIÓN Y SEGUIMIENTO

- La planificación y el seguimiento son esenciales para el éxito en proyectos de software.
  - La **planificación** define objetivos, tareas, recursos y cronogramas, alineando a todos los involucrados.
  - El **seguimiento** supervisa el progreso, detecta desviaciones y aplica medidas correctivas para cumplir plazos y metas.
  - Herramientas software, metodologías ágiles (Scrum, Kanban) y reuniones periódicas facilitan ambos procesos.
- Objetivos:
  - Convertir objetivos en trabajo ejecutable
  - Descomponer entregables y tareas
  - Estimar esfuerzo y duración
  - Asignar recursos
  - Establecer dependencias, hitos y calendario



# PLANIFICACIÓN: INCERTIDUMBRE

- La **incertidumbre es inherente** al desarrollo de software, por lo que las metodologías actuales (como las ágiles) no intentan eliminarla con planificaciones rígidas e irreales, sino que la integran como un factor natural.
- En lugar de doblegar el cambio artificialmente, adoptan ciclos cortos, retroalimentación continua y ajustes permanentes, convirtiendo la incertidumbre en una oportunidad para aprender y evolucionar el proyecto sobre la marcha.
- Aspectos a entender:
  - Una estimación no es una promesa
  - Cuanto más lejano es el trabajo, mayor incertidumbre
  - El plan se actualiza cuando cambia la información
  - Planificar  $\neq$  predecir perfectamente



# EJECUCIÓN Y CONTROL

- La ejecución y el control ya no se conciben como una mera verificación del cumplimiento del plan inicial, sino como un proceso dinámico de inspección y adaptación continua.
- Las metodologías actuales sustituyen los controles rígidos y puntuales por revisiones frecuentes (daily stand-ups, sprints revisiones) que permiten la detección temprana de desviaciones y reorientar el rumbo de forma colaborativa, aceptando que el plan es una guía abierta a reescritura constante según la realidad del proyecto.
- Se trata de:
  - Ejecutar el trabajo planificado
  - Medir progreso y resultados
  - Gestionar cambios, riesgos, problemas y decisiones
  - Comunicar continuamente a los stakeholders



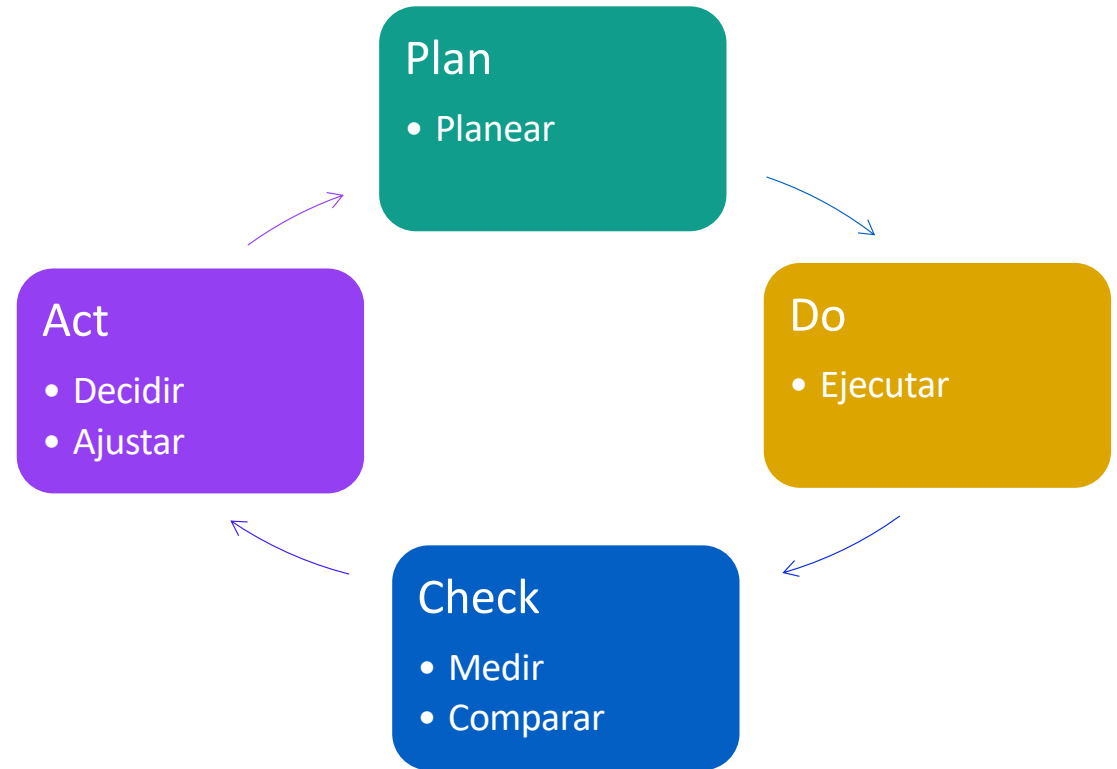
# CONTROL DEL PROYECTO

- El ciclo PDCA (Plan-Do-Check-Act) aplicado al control de proyectos convierte la gestión en un bucle de mejora continua: se planifica, se ejecuta, se verifica el resultado y se actúa para ajustar, asegurando que cada desviación detectada sea una oportunidad de aprendizaje y no un fracaso del plan.



# CONTROL DEL PROYECTO

- Resumen detallado:



- Preguntas clave:

- ¿Vamos tarde?
- ¿Qué ha cambiado?
- ¿Qué riesgo ha cambiado?
- ¿Qué decisión debemos tomar ahora?



# GESTIÓN DE PERSONAL EN PROYECTOS SOFTWARE COMPLEJOS

- El factor humano es el principal determinante del éxito o fracaso.
- Aspectos clave son:
  - **Dinámicas de equipo**
    - Formación, inboarding, normalización, desempeño y disolución (modelo Tuckman)
    - Equipos cohesionados y con confianza mutua son más productivos que la suma de individualidades
    - La diversidad cognitiva mejora la resolución de problemas, pero requiere tiempo de adaptación
  - **Conflictos**
    - No todos los conflictos son negativos: los conflictos de visión o tarea (sobre ideas) pueden ser creativos si se gestionan bien
    - Los conflictos personales o relacionales son destructivos y deben abordarse con prontitud
    - *El líder debe actuar como facilitador, no como juez*
  - **Comunicación**
    - La comunicación ineficaz es la principal causa de retrasos y errores
    - Canales claros, reuniones periódicas (daily, retrospectivas) y documentación justa y suficiente.
    - Comunicación asíncrona y síncrona: equilibrar para evitar saturación.



# GESTIÓN DE PERSONAL EN PROYECTOS SOFTWARE COMPLEJOS

- Aspectos clave (continuación):
  - **Tipos de personalidades (modelos útiles)**
    - MBTI / DISC / Belbin: conocer los perfiles ayuda a asignar roles acordes
    - Analíticos → detalle y calidad
    - Creativos → innovación y exploración
    - Ejecutores → cumplimiento y organización
    - Comunicadores → interacción con stakeholders
    - La diversidad de perfiles es una fortaleza, pero requiere que cada uno entienda y respete las diferencias del otro
  - **Influencia en el proyecto**
    - La motivación y el clima laboral afectan directamente la calidad del código y la velocidad de entrega
    - Equipos autoorganizados y con autonomía toman mejores decisiones técnicas
    - El agotamiento (burnout) en proyectos complejos es un riesgo real: hay que vigilar cargas de trabajo y celebrar logros
    - Conclusión clave
    - "Gestionar personas no es un soft skill menor, es el verdadero núcleo de la ingeniería de software. La técnica resuelve problemas, pero las personas los crean y los solucionan."



# CIERRE DEL PROYECTO

- El cierre del proyecto es una fase crítica que va más allá de entregar el producto, ya que incluye:
  - la verificación final de requisitos,
  - la documentación de lecciones aprendidas,
  - la liberación de recursos,
  - la transferencia a operaciones y ...
  - la celebración del logro,
- asegurando así que el proyecto se archive de forma ordenada y que su conocimiento perdure para futuras iniciativas.
- En caso de ciclos DevOps el cierre determina el fin del proceso de desarrollo inicial



# CIERRE DEL PROYECTO

- Se trata de:
  - Verificar aceptación de entregables
  - Cerrar contratos y documentación
  - Transferir conocimiento y operación
  - Registrar lecciones aprendidas
  - Liberar recursos



# MAPA GLOBAL PARA EL CURSO



- En todo el ciclo: stakeholders + riesgos + calidad + deuda técnica



# CASO TRANSVERSAL PARA LAS PRÁCTICAS

- Cada grupo definirá su propio proyecto
- El proyecto evolucionará durante el curso
- Cada práctica añadirá decisiones y artefactos
- Objetivo: aprender gestión gestionando un proyecto realista



## IDEAS CLAVE: ¡NO OLVIDAR!

- Gestionar un proyecto software es gestionar decisiones bajo incertidumbre
- El alcance debe ser explícito
- Los requisitos deben ser verificables y trazables
- La planificación debe adaptarse a nueva información
- Stakeholders y deuda técnica forman parte de la gestión durante todo el proceso



**PREGUNTAS**



# REFERENCIAS FUNDAMENTALES

- Project Management Institute (2021). A Guide to the Project Management Body of Knowledge (PMBOK® Guide), 7th ed.
- Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner's Approach, 9th ed.
- Sommerville, I. (2016). Software Engineering, 10th ed.
- Wiegers, K., & Beatty, J. (2013). Software Requirements, 3rd ed.



## REFERENCIAS: REQUISITOS Y AGILIDAD

- Cohn, M. (2004). User Stories Applied. Addison-Wesley.
- Leffingwell, D. (2011). Agile Software Requirements. Addison-Wesley.
- Kniberg, H., & Skarin, M. (2010). Kanban and Scrum.
- Christina Sookram. ¿Qué es el Marco Ágil Escalado (SAFe)? Disponible online:  
<https://thedigitalprojectmanager.com/es/gestion-de-proyectos/metodologia-safe/>



## REFERENCIAS: ARQUITECTURA Y DEUDA TÉCNICA

- Bass, L., Clements, P., & Kazman, R. (2021). Software Architecture in Practice, 4th ed.
- Nygard, M. T. (2018). Release It!, 2nd ed.
- Fowler, M. (2019). Technical Debt Quadrant. [martinfowler.com](http://martinfowler.com)
- Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical Debt: Towards a Crisper Definition and Better Dosing. IEEE Software.



## OTRAS FUENTES

- PMI — The Standard for Project Management
- IEEE — Software engineering standards
- IIBA — Requirements and business analysis
- Agile Manifesto — Principios de desarrollo ágil