Disclaimer: Este apunte no es autocontenido y fue pensado como un repaso de los conceptos, no para aprenderlos de aquí directamente.

Índice

1. Ingeniería de software

≠ programación:

- 1 desarrollador vs equipo
- sistema simple vs complejo
- tiempo corto vs largo
- desarrollador = usuario vs stakeholders
- poco mantenimiento vs muchisimo

≠ ciencias de la computación: parte práctica, no teórica. La ISW se nutra de CS.

No silver bullet: 4 dificultades esenciales

- Complejidad (no lineal con el tama no)
- Conformidad (arbitrariedad)
- Facilidad de cambios
- Invisibilidad

2. Modelos de ciclo de vida

Modelos de ciclo de vida: Representación estándar de etapas de desarrollo, orden relativo y criterios de transición.

Sirve para planificar, elegirlo es una decisión crítica.

Clave: **visibilidad**

Plan = modelo + parámetros (instanciarlo)

⇒ un tradeoff entre rapidez, documentación, calidad, visibilidad, riesgos, etc

- Cascada: Requerimientos, luego dise no, luego implementación (forward only)
- Cascada con prototipos: Igual, pero se van validando las etapas con prototipos. Problema: El prototipo debería tirarse por ser hecho sin pensar en calidad y nunca se tira.
- Sashimi: Cascada con superposición de etapas.
- Cascada con subproyectos: Minicascadas dentro. Soluciona un poco del problema (subproyectos mas cortos ⇒ el salto de salmón es mas corto).
- Iterativo (hacer varias veces lo mismo) e incremental (producto crece a medida que avanza el proyecto).
  - UP, RUP
  - SCRUM

Requisitos inestables / producto novedoso ⇒ iterativo
Arquitectura compleja ⇒ atacar riesgos desde el inicio
Se puede hacer D&C ⇒ iteraciones mas cortas
Complejidad en el negocio ⇒ cuidar especificación

3. Planificación

Gerenciamiento: Planificación, staffing, control, liderazgo, organización

Identificar stakeholders: Quién usa? Quién paga? Quién sabe? Quién quiere que exista? Quién es factor de decisión para que exista?

Claves: sponsor, lider usuario, usuarios directos e indirectos (gente afectada)

Driver: Algo extraído de la realidad que dirige el desarrollo. Qué tengo que intentar optimizar.
Restricción: Algo que restinge las posibilidades del proyecto.
Grado de libertad: Contrario a restricción.

Todos se aplican a: Funcionalidad - Calidad - Recursos - Costo - Plazo

Plan += alcance, requerimientos iniciales, QAs

Estimación: Mejora según avanza el proyecto

Falla: Optimismo, poca seriedad, no se recuerda bien la experiencia, novedad (falta de experiencia), mala administración de requerimientos

Estimo — Mido — Registro — Comparo — Analizo — Calibro — Estimo

¿Qué estimo? Tama no, esfuerzo, costo
Métodos de estimación:

- Algorítmico: Puntos de función, de objeto, de CU + factores de ajuste
- Empíricos: A ojo, basados en experiencia (proys similares)
- Wideband delphi (reunión convergente)

Work Breakdown Structures (WBS): D&C primero para estimar cosas mas chicas

- De proceso: Root=proyecto, nodos son tareas
- De producto: Root=producto, items de software, hardware y datos
- Híbrida: Proceso cerca del root, se torna producto llegando a las hojas

Dependencias entre tareas (no solo del tipo "fin a comienzo") determinan dinamicamente las fechas de las tareas. Hitos: Tareas de duración 0.
¡Hay dependencias hacia afuera del proyecto! con otros procesos de la empresa que tengan impacto (stakeholders)
Dependencias por contención de recursos

Gráficos de dependencias

- Gantt: Barras con fechas, respeta dependencias
- Pert/CPM: Grafo de dependencias, sirve para buscar camino crítico

Plan de gestión: Documento entregable

Línea de base: Versión freezada del plan, base para el seguimiento

Avance: Peso a hitos entregables ($\neq$ tiempo, $\neq$ esfuerzo). Valor acumulado.

4. Atributos de calidad (QA)

QA $\simeq$ reqs no funcionales, pero puede que se resuelvan con funcionalidad (ej, configuración)

QAs influyen fuertemente la arquitectura, que debe asegurarlos

QAs pueden contraponerse (ej: performance y flexibilidad)

Tácticas de arquitectura: Sirven para asegurar QAs (no son un fin en si mismas)

Especificación: Reqs (funcionales) + QAs + restricciones

Error $\Rightarrow$ defecto $\Rightarrow$ falla (observable por stakeholders)

Primer nivel de taxonomía de QAs

- Disponibilidad: Proba de estar disponible = TiempoMedioHastaFalla / (TiempoMedioHastaFalla + TiempoMedioReparación). Fácil de especificar, dificil de asegurar.
- Facilidad de cambios: ¿Qué puede cambiar? Funcionalidad, plataforma, otro QA, interfase. ¿Quién lo cambia? Usuario, desarrollador, administrador (configuración, código, parámetros)
- Performance: Latencia, deadline, throughput, jitter, #reqs no procesados. Difícil de expresar.
- Seguridad: Non-repudiation, confidencialidad, integridad, auditabilidad
- Facilidad de test: Test harness
- Usabilidad, escalabilidad, portabilidad

QAW (Quality Attributes Workshop): Método del SEI que relaciona a los stakeholders para detectar QAs clave. Brainstorming de escenarios (de ellos se deriva el resto).

5. Arquitecturas

5.1. Introducción

Arquitectura: Elementos + relación entre ellos + propiedades externamente visibles (interfase / minimo asumible por otros).

¿Por qué? D&C, plano sistema, comunicación con stakeholders y equipo, análisis temprano de QA

Arq vs dise no: Arq solo interfases. In the large.

Diagramas: Orientado a datos, C&C, cajas y flechas, capas, módulos, deployment.

Uso de la arquitectura

- Ingenieros de requerimientos: tradeoff entre reqs (QAs) en competencia
- Dise nadores: tradeoff contención de recursos y presupuesto
- Implementadores: Proveer restricciones y libertades (interfases)
- Testers/Integradores: Conocer comportamiento caja negra
- Soporte: análisis de impacto preliminar
- Dise nadores de otros sistemas: conocer interfase
- Managers: plano para asignar, planificar y seguimiento
- Grupo QA: análisis de conformidad
- Gestor de configuración: Organizar repositorios y SCM

Vistas: planos de la arquitectura, el sistema no es unidimensional

Clave: detallar vistas relevantes y **vincularlas**. La relevancia depende del propósito. Cada vista expone $\neq$ QAs.

*Ninguna vista es **la** arquitectura.*

Estilos: cliente/servidor, capas, datos compartidos, pipe&filter, publish&subscribe

Forma de abordar con propiedades conocidas. Se pueden usar $\neq$ en $\neq$ partes del sistema. Hay que documentarlos. Pueden ser desde la forma de encarar la documentación hasta una táctica.

6. Proceso Unificado (UP)

UML: Notación estándar (**no** es un proceso).

UP: Proceso "marco" $\rightarrow$ se adapta a las características particulares del proyecto.

- Dirigido por CU

- Centrado en arquitectura (prioritaria siempre, refinamiento progresivo)
- Iterativo e incremental, los riesgos determinan la construcción (administración de requerimientos)
- Iteración resulta en un sistema (feedback de los usuarios)
- Time boxing (si no se llega, recortar funcionalidad)

Tipos de iteración (fases)

- Inception: Establece **caso de negocio**. Especifica. Salidas: Caso de negocio, criterios de éxito, evaluación inicial de riesgos, estimación de recursos y requerimientos (10-20 %).
- Elaboration: Análisis, base arq, atacar principales riesgos, plan. Salidas: Modelo de dominio y CU (80 %). Arq testeada y documentada. Caso de negocio revisado. Plan de desarrollo.
- Construction: Especificación, desarrollo y test incremental.
- Transition: Desarrollo, test e implementación.

Hitos: Puntos de control para revisar el avance. Tiene entregables asociados
Principales: Fin de fase. Secundarios: Fin de iteración.

Disciplinas: Organizan actividades. De desarrollo: Requerimientos, análisis, arquitectura, dise no, implementación, test, deploy. De gestión: Riesgos, plan, seguimiento, SCM.
Generan modelo UML que incluye diagramas.

Artefactos: Información producida por el equipo (docs, código, etc).

Workflow: Especifica proceso en términos de actividades, artefactos y workers (con rol).

artefactos grandes ⇒ definir cuales se hacen en un desarrollo concreto.
artefacto inicial: caso de desarrollo. Decide justamente eso (para cada artefacto cuándo y dónde se crea y se actualiza).

Ejemplos de artefactos: Modelo CU, modelo de dominio, modelo de análisis, modelo de dise no, de arquitectura, de test, de implementación, prototipos, QA, glosario.

7. Gestión de riesgos

Riesgo: Problema que todavía no ocurrió <probabilidad de que ocurra,impacto>.
Proba * impacto = exposición al riesgo.

Identificar	**analizar**	**planificar**	**seguir**	**controlar**
		plan contingencia	cuantificar	ejecutar planes

Comunicar: Intercambiar info con todos para determinar tempranamente.

Identificar: id → doc → doc contexto (fuente, relaciones).
Métodos: Brainstorm, reporte periódico, cuestionario SEI, reportes voluntarios, lista de riesgos comunes.
¡Al avanzar el proyecto aparecen nuevos riesgos!

Documentación: Dado que [condición (fuente del riesgo)] ⇒ (posiblemente) [impacto]

Método de 3 niveles del SEI para priorizar la lista de riesgos. Probabilidad e impacto con 3 niveles, cada combinación tiene una prioridad.

Aproximación: Evitar, reducir probabilidad, reducir impacto (flowchart). Definir alcance y acciones de mitigación. Definir mecanismos de tracking (métricas que definen si el riesgo está ocurriendo).

Plan de contingencia: Medir impacto en el plan general. ¡Tiene que ser implementable!

Riesgos comunes: Producto incrrecto (mal reqs), producto incorrectamente (mal QAs), atrasos, costo muy alto.

8. Estimaciones

¡Siempre se puede estimar! aunque con cierto nivel de incertidumbre (baja a medida que avanza el proyecto) Mido a partir de la experiencia (mia o de otros, basado en similitudes varias). Es la basa para trabajar.

Problemas: Optimismo, falta de experiencia, omisión de tareas (si luego no se hacen comprometen la calidad, si se hacen comprometen el cronograma)

¿Qué estimo? Tama no, esfuerzo, costo, tiempo (problema: confundirlos!)

Métodos: Empíricos (ojo basado en experiencia), algorítmicos, descomposición (D&C). Combinación de varios.

Clark: $E = \frac{O+4M+P}{6}$. Elimina un poco sesgo optimista (la medición pesimista está mas lejos de la media que la optimista)

Wideband delphi: Muchos estimadores. Estimaciones sucesivas revelando progresivamente información para que converjan. Termina por tiempo o desvío aceptable.

Puntos de función: Contar entradas/salidas/consultas/archivos lógicos/ interfases. Ponderar por complejidad. Factores de ajuste (complejidades grandes)

Puntos de objeto (no es POO!): obj = pantallas, reportes y módulos (ponderados). Es mas simple y considera el reuso, pero solo sirve para ABM

Puntos de CU: Peso CUs y actores ponderadamente. Muchos factores de ajuste.

9. Seguimiento

Definición de cronogramas: División (D&C), dependencias, asignación de tiempo y esfuerzo (fechas cumplen dependencias), responsabilidades, salidas de cada tarea, hitos (chequeo de salidas)

Cronograma efectivo (Tomayko): Detalle de tareas *y recursos (+difícil)*, compatible con planes que interfieran, hitos claros con entregables (permite evaluar completitud - seguimiento).

Seguimiento: Proveer visibilidad a responsables del proyecto para reaccionar (relacion con riesgos). Mirar avance, esfuerzo, calidad (¿está **terminado**?)

Avance (valor acumulado): Peso a los hitos, subpeso a las subtareas que hacen el hito. Sumo peso de un hito tarea cuándo: está entregado, pasó SQA, fue aprobado por el cliente (puedo ir sumando porcentajes en cada uno de esos).

Esfuerzo: Detectar desvios ⇒ mejorar estimación. Dividir por fase/tarea. Reporte de horas del personal (difícil). Es importante para el **tracking de costo**.

También seguimiento de: riesgos, QA, proceso definido.

No funciona: Recuperar al final, eliminar QA o test, eliminar requerimientos sin acordar con el cliente, desatender integración, mucha sobrededicación. Se requiere asumir el problema de la estimación y corregirlo correctamente.

Administración de cambios: Hacerlos controladamente (importante en el SCM)

Especialmente administración de requerimientos. **van** a cambiar, estar preparado. Involucrar al usuario, comité de aprobación, analizar impacto, participan los que hicieron el análisis inicial (no olvidar cosas anteriores).

Management: Liderazgo, delegación.

10. Mejora de procesos (CMM)

Proceso de desarrollo:

IEEE: secuencia de pasos con un propósito. SEI: Conjunto de actividades, métodos, prácticas y transformaciones para dearrollar y mantener software.

Calidad de proceso ⇒ calidad de producto

Madurez de un proceso: Nivel de definición (calidad). Nivel de administración, control y efectividad.

Capacidad de proceso: Rango de resultados a partir de seguirlo.

"Software project performance": Resultado real. Su varianza (ruido) debe controlarse.

Inmadura: Procesos improvisados en cada proyecto, no hay control de procesos, más dependiente de las personas, menor visibilidad ($\Rightarrow$ menor control).
Madura: Procesos compartidos por todos, realistas (implementables!), actualizados continuamente, bien definidos, procesos sobreviven a las personas.

IDEAL: Initiating - Diagnosing - Establishing - Acting - Learning (forma de implementar cambios)

CMMi: Modelo, 5 niveles, cada nivel tiene temas que deben cumplirse para alcanzarlo. Se pueden usar prácticas de niveles superiores, pero deben tenerse todas para alcanzarlo.
Mejoras incrementales: Cada nivel sirve para llegar al otro.

CMMi continuo: Diferentes áreas, un número de valoración por área. Más flexible.

CMMi: Evolución del CMM (aprendizaje de a nos), más best-practices, mejores descripciones, cumple ISO 15504

Más madurez $\Rightarrow$ más visibilidad $\Rightarrow$ más capacidad de control $\Rightarrow$ menor ruido

Cada nivel tiene Key Process Areas (KPAs) que hay que cumplir. Cada KPA tiene objetivos y key practices organizadas en 5 secciones (common features).

SCAMPI (Standard CMMI Appraisal Method for Process Improvement)

- Clase A: (full) Foco en institucionalización. Otorga certificado.
- Clase B: Foco en deployment. Útil previo a ciertos procesos.
- Clase C: Foco en approach. Evalúa riesgos.

Niveles de CMMI

1. Inicial
2. Managed
3. Defined
4. Quantitatively managed
5. Optimizing