(1)

HOJA N°

FECHA

Problemas DSM software { Atraso proyectos, terminan, mueren, suspenden, congelan, postergan.
"Proyecto FASE II": (incompletitud) o reingeniería de.. (deuda técnica)
excede los costos, no cumple expectativas del cliente, no hace lo que debería
no calidad, problemas performance, seguridad, etc

CAUSAS – no administración o control
↳ "no se puede controlar lo que no se mide" (HAY MAS)

Reacciones? → eliminar etapas (testing, docs, etc)

Diagrama causa efecto

→ Util para recolectar datos, identificar causas principales
→ Nos da concepción del problema

SOFTWARE QUALITY: un SW es de CALIDAD cuando cumple los Requerimientos implícitos y explícitos del user
→ se discute en (todas las etapas) | costo de no calidad: -baja motivación del team -desconfianza -re-trabajo -img negativa (opinión juicio de expertos) (esta mal)

subjetivo { Visiones calidad → Trascendental: cumple solo las expectativas de la persona, lo cual es subjetivo
↳ Vision del user: definida por el cumplimiento de los requerimientos del usuario (funciones, etc) también es subjetivo
objetiva { ↳ Vision manufactura: calidad definida porque se respetó un X proceso de construcción, es decir, cumple un "estándar" → es: mc donalds (la burger se hace igual en todo el mundo)
↳ Vision producto: se dice que es de calidad cuando tiene cierto grado de cumplimiento con los atributos que fueron definidos. + características + calidad + costo
↳ Vision del valor: medida en la cant de dinero que el usuario está dispuesto a pagar (cuanto me cuesta) (costo-beneficio)

[Adecuación funcional]: ¿qué hace el producto? → si está completo\correcto funcionalmente, qué tan adecuado es para lo que yo necesito
↳ Completitud funcional: grado de cumplimiento de las funcionalidades (tareas + obj del user) ¿está completo?
↳ Corrección funcional: resultados correctos con el nivel de precisión requerido (capacidad del sw de "acertar")
↳ Pertinencia funcional: lo que me ofrecen es apropiado para lo que necesito?

[Eficiencia desempeño]: representa el desempeño relativo a la cantidad de recursos utilizados (performance)
↳ Comportamiento temporal: tiempos medios de rta\procesamiento desde un estímulo y su salida (benchmarks)
↳ Utilización de recursos: cantidad y tipo de recursos utilizados del software cuando está en funcionamiento
↳ Capacidad: grado en que los límites máximos de un parámetro de un SW cumplen con los requisitos

[Compatibilidad]: capacidad de 2 o + sistemas\componentes para intercambiar información o funciones cuando comparten el mismo entorno de SW o HW
↳ coexistencia: coexistir con otro SW sin problemas
↳ interoperabilidad: 2 o + sistemas\componentes para intercambiar info

[Usabilidad] → capacidad del SW para ser entendido\aprendido usado y resultar atractivo para el user
↳ accesibilidad, estética, protección errores, usable, aprendible, adecuación (ver si el sw cumple funcionalidad)

[Fiabilidad]: capacidad para desempeñar funciones específicas cuando se usa bajo unas condiciones y periodos de tiempo determinados
↳ madurez, disponibilidad, tolerancia fallos, cap. de recuperación [madurez: capacidad para satisfacer las necesidades de fiabilidad en condiciones normales]

[Seguridad]: capacidad de protección de la info\datos para que sistemas\users no puedan leer\modificar
↳ confidencialidad: que lo accedan los que deben → no repudio: poner logs para registrar acciones
↳ integridad: que la info no sea alterada → responsabilidad: rastrear unequívocamente las acciones de una persona
↳ autenticidad: demostrar la identidad de un sujeto\recurso

[Mantenibilidad]: capacidad para ser modificado efectiva\eficientemente.
↳ modularidad: capacidad de meter un componente (o cambio) y que ∃ poco impacto en los otros
↳ reusabilidad → analizabilidad → cap. para ser probado → cap. para ser modificado

[Portabilidad]: capacidad para ser transferido de forma efectiva\eficiente
↳ adaptabilidad: cuánta mano hay que meterle para pasarlo del entorno A → B
↳ cap. para ser instalado\reemplazado

PAPER CLASSIC MISTAKES: - Code-like-hell Programming: codean tan rápido como pueden → BUGS, MAL CÓDIGO

- EXCESIVE MULTITASKING: cuesta seguir el hilo al cambiar entre proyectos (pérdida de productividad)
- HEROICS: programador muy exp que soluciona todo siempre
- SILVER-BULLET-SYNDROME: bala de plata (que sea un new tech q resuelva todo)
- WISHFUL THINKING: no es optimismo, sino pensar q funcione a pesar q hagan las cosas mal

/PAPER/

GOOD ENOUGH: la calidad es negociable en función de las necesidades del user.

STANDING ON PRINCIPLE: no dejar que un PM afecte tu trabajo

/PAPER/

5 dimensiones → ALCANCE | TIEMPO | COSTO | CALIDAD | STAFF

¿x q lo hacemos?

3 Roles →
- DRIVER: objetivo a lograr (poca/nada flexibilidad)
- CONSTRAINT: limitaciones (poca/nada flexibilidad) → no tengo control sobre eso
- DEGREE OF FREEDOM: grados de libertad (flexible) → libertad para fijar objetivos

- siempre diálogo entre Devs + PM y PM + customer
- el PM debe poder categorizar las dimensiones para un orden (?)

CMMI: CAPABILITY MATURITY MODEL INTEGRATED → es un marco que nos permite construir procesos y se basa en (marcos) acciones con buenas prácticas)

- No es un proceso para desarrollar SW ni tampoco una metodología | determina la madurez de un proyecto

MADUREZ: capacidad organizacional de cumplir sistemáticamente con los objetivos

ENFOQUE CONTINUO: máxima flexibilidad ENFOQUE ESCALONADO: marca un camino de mejora

5 NIVELES MADUREZ
- ↳ CAÓTICO: todas las org asumen que están acá al empezar a trabajar como construir el SW
- ↳ REPETIBLE: empieza a definir q prácticas serán recomendables (gestión de costos)
- ↳ DEFINIDO: los distintos proyectos trabajan de igual forma
- ↳ ADMINISTRADO-CUANTIFICADO: empieza a medir todos los trabajos que voy realizando
- ↳ OPTIMIZACIÓN: retroalimentando feedback con buenas prácticas (innovar, nuevas techs, etc)

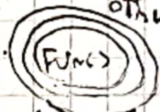
LEAN: enfoque q hace hincapié en la eliminación de residuos o de no valor añadido al trabajo

→ se aplica en: RRHH, IT, ventas, etc (ej. demasiadas reuniones)

- el diagrama causa-efecto es formado x Lean

TIMEBOX: técnica de scrum para limitar la duración de los eventos, como podría ser construir un SW.

→ construcción como cebolla:



others

| FUNCS en el centro y otras en las capas (es usado en desarrollo, de ahí timebox development)

→ se desarrolla lo + importante primero

(SE USA DE 60 A 120 DÍAS)

CARACTERÍSTICAS:
1. Prioridad en la planificación q queda en el tiempo estimado
2. Evita problema 90-90: cuando el proyecto está al 90% pero se queda ahí x meses
3. Hace en evidencia las funcs principales
4. Limita la perfección del desarrollador: tienes acceso → hago lo mejor posible más rápido
5. Controla funcs raras
6. Motiva devs/usuarios: por el sentimiento de "urgencia"
7. Mucha participación del final user

BOK: body of knowledge - requisito para calificar a cualquier área de la ingeniería como profesión define el conocimiento relevante para x disciplina

SWEBOK: software engineering body of knowledge

- ↳ software requirements
- ↳ software design
- ↳ software construction
- ↳ software testing
- ↳ software maintenance

NOTA

(2)

Roles de un Proyecto → Stakeholders: todos los involucrados, tienen poder de decisión y pueden influir sobre el proyecto

→ Sponsor: el "owner", es quien quiere que se lleve a cabo

→ User champion: experto en el dominio del problema del proyecto. Sabe como funciona todo el negocio y las funcionalidades del SW (muy demandado)

→ User directo: operan el sistema

→ User indirecto: hacen uso pero no lo operan

SYNEFIN → compara carac. de 5 dominios distintos (simple, complicado, complejo, caótico y desorden)

SCRUM → Divide el = {Alcance (entregables pequeños) / org (teams pequeños) / tiempo (ciclos)}

→ Optimiza: {el proceso (retrospectivas), proyecto (revisión) entregables y planificaciones}

Valores de scrum: (Focus, commitment, respect, courage, openness)

Roles → Scrum Master: dueño del proceso, colabora con team, ayuda a rojos y enseña (elimina trabas)

→ Product owner: dueño de la def de éxito, representa al cliente/usuario, (dueño del Backlog) es como el user champion, planifica el largo plazo. Define prioridades, cuida el valor del producto, asegura que el team labure

→ Equipo: Verifica entregables (el scrum tiene su parte)

Sprint Review: al fin del sprint ⇒ revisan el incremento del proyecto y ven si lo que hicimos se adecua a lo esperado

Retrospective: revisar la forma de trabajo del últ sprint para la mejora continua

ESTIMAR → Reqs (que tengo q hacer) → tamaño → esfuerzo (horas ganadas en el tamaño) → duración (cuanto dura mi esfuerzo)

¿xq fallan las estimaciones? → (cuando te piden el tamaño JS) → optimismo / falta exp / complejidad → no hay historia / → no se estima el tamaño

¿Tips? → tener en cuenta todo (testing, doc, código) → obtener datos históricos

Ciclo estimación: (Estimar (esfuerzo + costo)) → medir → registrar → analizar → calibrar → (volver a estimar) mínimos + se repite ++ mejoras +++ fit realidad/estimation

Tienen nada ni técnica, usan historia ← **Métodos rudimentarios** → "Juicio Expertos": el único input es la experiencia

→ Pert/Clark: busca normalizar el JE ↑ (eg. est = (optimista + 4 × medio + pesimista) / 6)

→ Wideband Delphi: juicio de expertos en grupo (usado cuando no ∃ historia)

→ Planning Poker: ya sabes (tampoco cuenta con historia) (usado en etapas tempranas)

(métodos sin fórmula que estiman el **esfuerzo**)

Usan historia ← **Métodos paramétricos** → Function Points: miden el tamaño del SW en base a la funcionalidad

→ Use Case Points: cantidad/complejidad de los uses cases, transacciones, basados en FP. Se fija en ⇒ actores simples (sist externo), actores promedio, actores complejos (humanos, impredecibles)

→ Story Points: (como en el laburo) miden complejidad, cantidad, riesgo e incertidumbre

→ Object Points: usado en proyectos de mantenimiento de SW {objeto ⇒ pantalla, reportes, módulos}

- se asigna a c/componente un peso según su complejidad (simple, medio, difícil)
- luego se brinda un peso a c/nivel y la Σ c/objetos con su peso = OPS

(métodos que estiman el **tamaño**)

Escenarios: Simple: tenemos las opciones y se como resolverla, usamos las mejores prácticas, elegimos la mejor alternativa → (sense - categorize - respond)

Complicado: se cual es mi objetivo, pero necesito un experto q me diga la mejor opción entre varias, requiere exp o análisis (tamb se aplican buenas prácticas) → (sense - analyze - respond)

Complejo: ∄ respuestas correctas, hay soluciones adaptativas, requiere creatividad, innovación, interacción y comunicación. Se aplican prácticas emergentes → (probe - sense - respond) ∄ tal experto xq acá pasan cosas que no se han realizado antes

Caótico: no tengo idea como salir y voy probando iteraciones hasta salir. Hay incertidumbre, distintas opciones que no sabes como van a salir → (act - sense - respond)

Desordenado: no sabemos ni en que dominio estamos, debemos tratar de ir a una situación que si sepamos como actuar.

KANBAN: es una disciplina donde podes ver visualmente como el flujo de trabajo funciona, sirve para ver donde funciona bien, donde no, donde mejorar, donde ∃ bottlenecks

→ usado cuando el proyecto no es estable y las tareas van cambiando de prioridad, tmb cuando los reqs van cambiando, o en entornos de resolución de incidencia

SCM: SOFTWARE CONFIGURATION MANAGEMENT (vos definís eso, para ir un sw de tareas, casos de prueba, etc)

[¿que es el SW? — Item de config: {cualquier elemento involucrado en el desarrollo del proyecto (codigo, DB, servers, scripts, etc) → se conoce nombre/version/rama

CONFIG de SW: conjunto de todos los componentes compilados en un ejecutable

SCM: Disciplina q administra la evolucion de {Productos → entregables; Procesos → para desarrollar productos/entregables; Ambientes → donde se ejecutan los procesos}

su proposito es establecer INTEGRIDAD de los productos del proyecto SW a lo largo de su life cycle

Todas las partes del sw se mantienen integras → garantiza la gestion de config.

BASELINE: Foto de la config. pto de referencia | Branch. (literal el concepto de GIT-Branch)

0. Admin. de SCM: etapa previa al arranque, se trata de comprender el contexto de la ORG

- ver la config, que herramientas hay, existe algun SCM? Un plan SCM define: → listas de items de config → estandares nombres, directorios jerarquia → policys de branching/merging → builds, releases (como

1. Ident. la config: definir elementos que estaran controlados por la gestion de config, es decir, armar la hoja

2. Control de la config: Aseguran q los items de config mantienen su INTEGRIDAD, se establece un procedimiento de control de cambios. (eg. new branch → PR → merge/deploy) → etapa + formal (es uno linux o windows)

3. Status & Accounting de la config: registran y reportan info para admin la config. (es literal usar GIT) → listan PRs aprobados → mostrar estado de cambios → trazabilidad de los cambios

4. Admin de distribucion/despliegue SW: tenemos el codigo ready los cambios → "Deployar + Produccion". Consiste en liberar el SW a otros entornos (dev, stage, prod) se divide en 2 → Software Building; → Release Management (version, variante, release)

Tipos de builds — LOCAL BUILDS: en la PC del dev

↳ INTEGRATION BUILDS: genera entorno completo para prueba de integracion

↳ NIGHTLY BUILDS: ejecuta construccion diariamente y genera reportes de estabilidad, tiempo, etc

↳ RELEASE BUILDS: disparados cuando un admin decide crear una nueva version

"Todos los builds deben ser automaticos, generan reportes" → generado el build ahora viene el deploy

BUILD Y DEPLOYMENT PIPELINES: camino completo desde que se compila el sw seguido del progreso a traves de varias etapas de testing y deployment. Es la manifestacion automatizada del proceso para llevar el software desde la aprobacion del SCM hasta que llega a los usuarios

→ BASIC DEPLOYMENT: todos los nodos de la app son actualizados en el mismo momento → simple pero puede generar downtime

→ BLUEGREEN DEPLOYMENT: dos ambientes similares, en uno se hace el deploy y el otro contiene la version anterior → es costoso, garantiza HA

→ ROLLING DEPLOYMENT: se actualizan los nodos 1 a 1 (en lotes) con la nueva version del sw, aunque el rollback es simple, debe haber consistencia para trabajar con diferentes versiones

→ CANARY DEPLOYMENT: deployan en un set controlado de nodos e ir chequeando el behavior, me muevo de a porcentajes (2% de los nodos). solo existe 1 Amb de produccion, lo cual es bueno pero garantiza complejidad.

5. Auditoria de la config: realizan una verificacion del estado de la config para ver si se estan cumpliendo los reqs. especificados (checklists o pruebas exhaustivas)

Puede ser → funcional (valida funciones por reqs), fisica (estructura, defectos del codigo) o de proceso (verifica que se cumplio el proceso de SCM)

CD (continous delivery): practicas que permiten asegurar que el SW puede ser desplegado en produccion rapidamente y en cualquier momento.

CI (continous integration): integracion de codigo al menos 1 vez x dia (asi minimizas problemas)

Good practices: {repo de codigo unico, build automatico, pre-commit, build rapido, deployment automatico}

Delivery: todos los pasos son automatizados excepto el deploy

Deployment: todo es automatizado

DevOps: conjunto de practicas destinadas a reducir el tiempo entre el cambio en un sistema y su pasaje a produccion. combinacion de desarrollo + operaciones

TESTING

"CONOCIMIENTO": CAJA NEGRA\BLANCA\GRIS | "TIPO DE PRUEBA": FUNCIONALES\NO FUNCIONALES
"EJECUCIÓN": AUTOMATIZADAS, MANUALES, SEMI | "ALCANCE": UNITARIAS, INTEGRACIÓN, ACEPTACIÓN DE USUARIO

SOFTWARE TESTING: DEFINIDA COMO UNA ACTIVIDAD PARA EVALUAR SI LOS RESULTADOS OBTENIDOS EN LA EJECUCIÓN COINCIDEN CON LO ESPERADO, PARA DETERMINAR SI EL SISTEMA ESTÁ LIBRE DE DEFECTOS. LAS PRUEBAS ENCUENTRAN FALLAS, NO DEFECTOS.

¿OBJETIVO del TESTING? ⇒ ENCONTRAR FALLAS EN EL PRODUCTO. → {EFICIENTE: RÁPIDO Y BARATO; EFICAZ: ENCONTRAR LA MAYOR CANT. DE FALLAS, LAS MÁS IMPORTANTES Y NO DETECTAR FALSOS POSITIVOS}

NUNCA HAY CERTEZA DE CUÁNTAS FALLAS TOTALES HAY → FALSOS POSITIVOS.

¿NUEVA DEF. DE SW DE CALIDAD? ⇒ AL SER USADO, NO CONTENGA FALLAS + CUMPLA REQUISITOS

→ (UNA PRUEBA ES EXITOSA SI ⇒ ENCUENTRA FALLAS.)

EQUIVOCACIÓN: ACCIÓN HUMANA Q' PRODUCE UN RESULTADO INCORRECTO

DEFECTO: PASO, PROCESO O DEFINICIÓN DE DATO INCORRECTO (AUSENCIA DE CIERTA CARACTERÍSTICA)

FALLA: RESULTADO DE EJECUCIÓN INCORRECTA, PRODUCIDO POR EL SW (ES LA MANIFESTACIÓN DE 1 O MÁS DEFECTOS)

CONDICIONES DE PRUEBA: DESCRIPCIÓN DE SITUACIONES Q' QUIEREN PROBAR (ES UN SW QUE SACA EL PERÍMETRO, NO ME DEBERÍA DEJAR PASAR ALGO CON UN LADO = 0)

CASOS DE PRUEBA: LOTES DE DATOS PARA QUE SE CORRA UNA CP

(INCIDENTES EN TESTING): OCURRENCIA DE UN EVENTO QUE OCURRE DURANTE UN TEST DE SW, NO TODO INCIDENTE ES UNA FALLA, PUEDE PASAR POR PROBLEMAS EN EL ENV POR EJEMPLO

(DEPURACIÓN): ELIMINAR UN DEFECTO DEL SW (DETECTAR → DEPURAR → VOLVER A PROBAR → APRENDER)

(¿CUÁNDO PARO DE PROBAR, O HACER TESTING?) → CANT FALLAS DETECTADAS ≅ CANT FALLAS ESTIMADAS
→ "GOOD ENOUGH": CIERTA CANT. DE FALLAS NO CRÍTICAS ES ACEPTABLE

PRUEBAS FUNCIONALES

ENFOQUE CAJA NEGRA: PRUEBA LO QUE EL SW DEBERÍA HACER {SELECCIONAMOS UN CONJUNTO DE DATOS DE ENTRADA Y A C/ SUBCONJUNTO LO LLAMAMOS "CLASE DE EQUIVALENCIA"}

PARTICIÓN DE EQUIV.
↳ IDENT. LAS CLASES
↳ DEFINIR CASOS DE PRUEBA

BORDER CASES: ES EDAD ≥ 18

BC INVÁLIDO: 17 — BC VÁLIDO: 18 — 16 17 | 18 19 20 … 80 (INVÁLIDO | CUALQ. VÁLIDO)

(EXPLORAR BCS → MEJORES RESULTADOS)

• EN LAS CLASES INVÁLIDAS → TMB PROBAR COSAS DE OTRO TIPO, COMO "/-#N.." ETC O COMBINAR "PROBAR ENTRADAS ENTRE INVÁLIDOS"

CONJETURA ERRORES ⇒ "PRUEBA DE SOSPECHAS" → ENUMERAR LISTA DE ERRORES O SITUACIONES PROPENSAS A TENER ERRORES

ENFOQUE CAJA BLANCA: PRUEBA LO QUE EL SW HACE → (TODA LA ESTRUCTURA INTERNA ES CONOCIDA POR EL TESTER)

EL TESTER SABE CÓMO ES EL CÓDIGO DE UNA PRUEBA, PODRÍA PROBAR "QUÉ PASA SI SE CAE LA DB", "QUÉ PASA SI ME PASO DE UN LÍMITE"

GRADOS DE COBERTURA:

- **COBERTURA DE SENTENCIAS**: PRUEBA C/ INSTRUCCIÓN, LÍNEAS DE CÓDIGO (COVERAGE)
- **COBERTURA DE DECISIONES**: PRUEBA C/ SALIDA DE UN "IF", "WHILE", "SWITCH"

→ NO SE PUEDE TENER COBERTURA DE SENTENCIA AL 100% Y COBERTURA DE DECISIONES XQ (EJ) PASAR POR UNO DE LOS LADOS DEL IF Y TENGO TODA ESA PARTE DEL CÓDIGO PINTADO. SENTENCIAS VA A ESTAR AL 100%, PERO DECISIONES AL 50%.

- **COBERTURA DE CONDICIONES**: PRUEBA C/ EXPRESIÓN LÓGICA (A and B || C=D) DE LOS IF\WHILE EN ESE IF TENGO 4 SITUACIONES Y 2 CAMINOS.
- **PRUEBA DEL CAMINO BÁSICO\COMPLEJIDAD CICLOMÁTICA**: PRUEBA TODOS LOS CAMINOS INDEPENDIENTES. LO IDEAL ES COMPL = 1 (PLANO, SECUENCIAL, SIN IFS NI NADA)

AMBOS CASOS SE COMPLEMENTAN → ∃ DEFECTOS IMPOSIBLES DE DETECTAR EN CADA MÉTODO

(TIPOS DE PRUEBA) → {PRUEBA UNITARIA, PRUEBA DE INTEGRACIÓN, PRUEBA DE ACEPTACIÓN DE USER}

PRUEBAS NO FUNCIONALES

: PRUEBAN REQUERIMIENTOS NO FUNCIONALES

- **PERFORMANCE**: QUE EL SISTEMA SOPORTA LOS TIEMPOS DE RTA DEFINIDOS → IDENTIFICAR BOTTLENECKS
- **STRESS**: SOMETER AL SISTEMA PASANDO SUS LÍMITES DE PROCESAMIENTO\ALMACENAMIENTO
- **SEGURIDAD**: PRUEBAN SI PUEDE SER VULNERADO (PENETRATION TEST, AUTH, ETC)
- **USABILIDAD**: PRUEBA ATRIBUTOS DE USABILIDAD (ERRORES + FRECUENTES, ELEMENTOS NO USER-FRIENDLY, ETC)
- **VOLUMEN**: PARA VERIFICAR Q' EL SISTEMA SOPORTA LOS VOLÚMENES DEFINIDOS (ALMAC., PROCESAM., TRANSMISIÓN)
- **REGRESIÓN**: VERIFICA Q' LUEGO DE METER CÓDIGO NUEVO, LA FUNCIONALIDAD ORIGINAL SIGUE IGUAL
- **SMOKE-TEST**: NO SE ENTRA EN DETALLES, PRUEBA BÁSICA Y RÁPIDA
- **α Y β**: LANZAR VERSIONES BETA PARA EL USER Y α PARA PROBARLO ...

EXPLORATORY TESTING: APRENDIZAJE, DISEÑO Y EJECUCIÓN DE LA PRUEBA EN FORMA SIMULTÁNEA. ES "UNSCRIPTED" (SIN GUIÓN) EL TESTER VA DECIDIENDO TÁCTICAMENTE TODO.

(SCRIPT DRIVEN ≠ UNSCRIPTED) → NO GUIONADO