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

1. Metodologías Ágiles

Las metodologías ágiles cambian el foco respecto de las tradicionales:

Individuos y sus interacciones	>	herramientas + procesos
Construir software	>	documentación completa
Colaboración con el cliente	>	negociación de contratos
Responder ante el cambio	>	el seguimiento del plan

Principios básicos:

- Visibilidad
- Inspección (detectar desvios a tiempo)
- Adaptación

Otras características: Empíricas, "livianas", adaptativas, rápidas pero no apuradas, emergentes (!), exponen pérdidas, centradas en el cliente, decisiones en niveles bajos, self-organized, time boxed

Se asume que va a haber cambios, se toma el mismo como forma de acercarse al deseo del cliente. Igualmente tiene mecanismos de control y estructura.

No sirven en equipos grandes o distribuidos (baja comunicación), si hay poca experiencia, en donde todo ya anda bien y cuando es dificil definir tarea terminada.

1.1. SCRUM

Pasos de SCRUM:

1. Product backlog: Conjunto de items que son user stories (reqs func y no func). Es lo que se sabe que se quiere que este en el producto final.
2. Sprint planning: Seleccionar subconjunto del product backlog según priorización del cliente y dividir los items en tareas según cree el team.
3. Sprint backlog: Items + taskboard
4. SPRINT (daily standups): Personas se autoasignan tareas del taskboard y las hacen. Daily standups: Cerditos dicen lo que hicieron, los pollitos miran. Sirve para dar comunicación y que todos vean la big picture (las cosas se retrasan un dia por vez por falta de comunicación). El taskboard se actualiza todos los dias con estimación de tiempo de tareas (1-16 hs). Tareas largas hacen daily standups improductivos (todos dicen "lo mismo que ayer").
5. Producto incrementado
6. Sprint review: Con el cliente, managment y product owner. Feedback.
7. Sprint retrospective: Sin el cliente, mejorar el proceso.

Roles de SCRUM:

- Scrum master: Usa la lista de impediments para resolverlos. Asegura el cumplimiento del proceso.
- Product owner: "Abogado" del cliente. Su objetivo es maximizar ROI. Prioriza los reqs, resume la input del cliente. Puede cambiar reqs entre sprints, puede aceptar/rechazar.
- Team: Los que hacen. No hay roles, solo skills (desarrollo, testing, analista funcional). Estima y define sprint.

Inicio: Product backlog con items iniciales, funcionalidad y #sprints hasta el release, evaluación de riesgos, capacitiación de la gente (incluido cliente) en SCRUM.
Fin: Cierre release, instalación, documentación y entrenamiento.

Especificación: Epics (grupos de funcionalidad), user stories (detalle de epics) que incluyen CU y casos de aceptación.

1.2. eXtreme Programming

Roles vs responsabilidades: Cualquiera podría tomar cualquier rol: Desarrollador, cliente, tester, tracker, consultor, coach, manager.

Valores principales: Comunidad, simplicidad, feedback, coraje y respeto.

Prácticas:

- Planning process
- Releases chicos, iteraciones cortas
- Metáforas (lenguaje común con el cliente)
- Dise no simple.
- Testing continuo como parte integral del desarrollo
- Refactoring
- TDD (test-driven development): Escribir test antes de desarrollar, correrlo para que falle, arreglarlo.
- Pair programming ⇒ Collective ownership (rotar pares y roles dentro del par)
- Collective ownership
- Integración continua
- Semana de 40 horas
- Cliente on-site
- Estándar de código

Ciclo dorado:
requerimientos cambiantes ⇒ dise no emergente ⇒ refactoring ⇒ testing continuo ⇒ pair programming ⇒ collective ownership ⇒ on-site client

	Exploration	Commitment	Steering
Release planning	stories/priorizar	alcance release	ajustes +/- reqs
Iteration planning	task cards	asignar tasks, estimación	iteración de trabajo y contraste con stories

2. Software Configuration Managment (SCM)

Control de cambios: Custodiar integridad y acompa nar los cambios con control.

Incremento en complejidad:
Soft: Multiplataforma, > tama no, interfases, OTS
Env: Tama no, distribución geográfica del team, frecuentes releases, cambios SO/hard.

1. Id y almacenar artefactos en repositorio seguro
2. Controlar y auditar cambios en artefactos (aumenta el control al avanzar el proy)
3. Organizar artefactos en SCM components:
   - Contribuye a preservar integridad
   - Reduce complejidad (D&C)
   - Facilita determinación del nivel de calidad
   - Aumenta el código compartido y el reuso
4. Crear baseline para cada milestone del proyecto
   - + Baselines al final de iteración o etapa de entrega
   - + Reproducibilidad (versiones). Id problemas introducidos en releases nuevos.
   - + Traceability (unif. artefactos)
   - Reporting / logging (facilita debug y notas de release, diffs)
5. Registro y seguimiento de pedidos de cambio (change request managment)
6. Mantener entorno de trabajo estable
7. Soportar cambios concurrentes sobre artefactos
8. Integración en forma temprana y frecuente
9. Asegurar que es posible reproducir releases (mantenimiento)

Integración: Merge o assembly.

Consistencia: Turn-taking, split-merge o copy-merge.

3. Calidad

Definiciones:

- Aptitud para uso
- Ausencia de defectos
- Satisfacción de requerimientos
- Nivel hasta el que un producto tiene un conjunto de atributos

V&V:
Validación: Producto correcto (requerimientos correctos) Verificación: Código correcto respecto de los requerimientos

Error (humano) ⇒ Defecto (en el código) ⇒ Falla (observable)

Modelo en V:

Requerimientos de sistema	Integración del sistema
Requerimientos de soft	Test de aceptación
Dise no arquitectura	Integración de software
Dise no detallado	Test de componentes
Implementación en código	Test de unidad

Testing: El crear tests ya ayuda por el proceso mental de pensarlos.
¿Cuándo termina? Heurísticas usuales: No hay fallas, porcentaje aceptable de fallas, # defectos hallados $\sim$ # defectos estimados, error seeding.

Tipos de test:

- De unidad
- De integración
- De sistema
- De aceptación: Baasado en requerimientos, dise nado y ejecutad por usuarios
- De volúmen y performance (borde)
- De stress (exceso)
- De regresión (luego de un cambio, ver que no se introdujeron fallas)
- α (usuario interno) β (usuario real)
- De usabilidad
- Funcional: Tipos de fallas, sincronizados con la especificación que les dió origen.

Revisiones

- Walkthrough: Presentador (conoce a fondo el producto) + asistentes
- Revisiones (formalidad intermedia)
- Inspecciones de código (mayor formalidad): Lector, revisor, autor, registrador y moderador. Obtiene como resultado de cada error: ubicación, descripción, severidad y tipo (interfaz, datos, lógica).
  Objetivos secundarios:
  - Aumentar visibilidad
  - Buscar consenso
  - Favorecer trabajo en equipo
  - Obtener datos para las métricas
  - Lector, revisor,

  Hay que hacerla cuando el testing es intermedio, al principio hay demasiados bugs y luego de testing exhaustivo no hay lugar para el trabajo.

4. Métricas

Objetivo previamente esclarecido:
Identificar objetos del dominio ⇒ definir escala (relacionada con los objetos)

¿Para qué usar métricas?

- Extender y modelar procesos y productos
  - Comparar lineas de base ($\neq$ versiones)
  - Predecir el efecto de un cambio
  - Calcular esfuerzo/recursos/tipos de error comunes

- Administración: Estimación, detectar desvios, averiguar relación entre parámetros (ej: erroes/linea de código)

Las métricas pueden ser de producto, de proyecto o de proceso.

4.1. Goal Question Metric (GQM)

Goal: Parte conceptual, objetivo.
Question: Parte operacional, modelo.
Metric: Parte cuantitativa, métrica.

Construir software ⇒ proceso de ingeniería ⇒ retroalimentación y evaluación ⇒ objetivos medibles dirigidos por un modelo.

El objetivo depende de la perspectiva (cliente ≠ proyeto ≠ corporación).

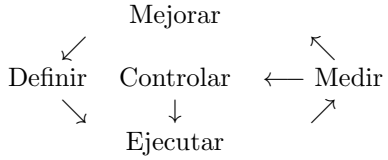
1. Definir objetivo de proyecto y corporativo: Operacionalmente refinado con preguntas medibles.
2. ¿Qué datos "hablan" sobre ese objetivo?
3. Definir métricas de respuesta y universo de interpretación
4. Medir
5. Interpretar

Fallas: Medir demasiado (medir indicadores irrelevantes), usar resultados para premio o castigo, cortoplazismo, esfuerzo de medición no homogéneo, no usar el resultado de las mediciones.

4.2. Alta madurez

Alta madurez: Gestión cuantitativa de procesos y productos.

Según CMMI: Uso sistemático de mediciones de los procesos para predecir costos, cronograma y defectos y establecer un rango de performance.



Definiciones de procesos:

- Performance (del proceso): Calidad, costo, tiempo
- Capacidad: Rangos de atributos
- Control cuantitativo (no estadístico)
- Control estadístico
- Estabilidad: Predicabilidad

CMMI N4:

- Baseline de performance y modelos
- Manejar cuantitativamente
- Manejar performance de subprocesos estadísticamente

Saber como se comporta el proceso con números: % de tiempo de gestión. # de bugs en un peer review. ⇒ Planificar (qué, cómo y cuándo medir) ⇒ Acumular números de muchos proyectos ⇒ Predecir estadísticamente.

Trabajo: Serie de procesos interconectados, todos pueden fallar.

- Entender pasado cuantitativamente
- Controlar presente cuantitativamente
- Predecir futuro cuantitativamente

Causas: comunes (esperables) vs especiales (no esperables) de variación.

Mediciones a varios niveles: Empresa, línea de producto, proyecto, proceso, subproceso. Mas alto, menos preciso el análisis.

Control estadístico: Problemas: Muy dificil distinguir causas comunes de especiales. Si o si tienen que ser proyectos homogéneos (tecnología, tama no).

SPC

- Elegir procesos adecuadamente, no combinar datos de organizaciones $\neq$
- Ajustar límites
- Efecto hawthorne (el medir perturba, incertidumbre)
- Comportamiento disfuncional (se trabaja para la métrica)

CMMI N5: Pasar de detectar a prevenir defectos. Mejorar segun entendimiento cuantitativo.

- Detectar causas de defectos $\Rightarrow$ atacarlas
- Seleccionar mejores $\Rightarrow$ implementarlas (incluye medir sus efectos)

Al analizar causas de defectos: Identificar, clasificar y priorizar. Identificar efecto y sus posibles soluciones. Prevenir nueva aparición (hacer pilotos con objetivos de performance y medirlos, si andan, extender).

Alta madurez en CMMI: N4: Comprensión y administración de variación de procesos para obtener calidad. N5: Mejorar tecnologías innovadoras. Implementarlas utilizando conocimiento cuantitativo.

5. Architecture Trade-off Analysis Method (ATAM)

Utilidad: Valoración del stakeholder (SH)
Valor: Caracterización del impacto de las decisiones

Beneficios:

- Fuerza la preparación de material para review
- Captura motivaciones de la arquitectura
- Determinación temprana de problemas
- Valida requerimientos
- Mejora arquitectura

Dificultad: Muchos SH $\Rightarrow$ diferencias valoraciones de utilidad $\Rightarrow$ vínculo decisiones de dise no - visibilidad SH

Precondiciones:

- Requerimientos y objetivos de arquitectura articulados
- Alcance definido
- Cost-efectiveness
- Miembros clave disponibles (SH de la arquitectura y project-decision makers)
- Team de evaluación competente (3 a 5 externos, parte de QA o ad-hoc)
- Expectativas gestionadas

Fases:
0. Contratos y NDAs, información inicial requerida (introducción)
1. Evaluación. Team + decision makers (uniformizar)
2. Evaluación. Team + decision makers + SH (consenso)
3. Segumiento, emisión de resultados, analisis post-mortem

Árbol de utilidad: Tiene objetivos de calidad en los nodos y escenarios en las hojas.

1. Presentar ATAM y árboles de utilidad
2. Presentar drivers de negocio: Requerimientos funcionales y de calidad
3. Presentar arquitectura: Restricciones técnicas, otros sistemas para interactuar
4. Identificar enfoques de la arquitectura: Aspectos clave QA
5. Generar árbol de utilidad de atributos de calidad.
6. Analizar enfoques de la arquitectura: Identificar enfoques QA prioritarios, generar preguntas para QA prioritarios, identificar riesgos, no-riesgos, puntos sensibles y compromisos.
7. Brainstorming: Priorizar escenarios. Agregar escenarios al árbol inicial (los iniciales sirven de ejemplo)
8. Analizar enfoques de arquitectura sobre lo surgido en el punto anterior
9. Presentar resultados. Salidas:
   - Presentación concisa de arquitectura
   - Articulación de los objetivos del negocio
   - Requerimientos de calidad expresados en escenarios

- Mapeo de requerimientos de calidad con decisiones arquitectónicas
- Identificación de puntos sensibles, trade-offs, risks y non-risks (asunciones)
- Risk themes

$\longrightarrow$	Drivers	$\rightarrow$	QA	$\rightarrow$	Escenarios	$\longrightarrow$	Analisis
$\longrightarrow$	Arquitectura	$\rightarrow$	Approach	$\rightarrow$	Decisiones	$\longrightarrow$	$\downarrow$
$\uparrow$							$\downarrow$
$\uparrow$					Trade-offs	$\longleftarrow$	$\downarrow$
$\uparrow$					Puntos sensibles	$\longleftarrow$	$\downarrow$
$\uparrow$					No-riesgos	$\longleftarrow$	$\downarrow$
$\nwarrow$	$\longleftarrow$	$\leftarrow$	$\longleftarrow$	$\leftarrow$	Riesgos	$\longleftarrow$	$\downarrow$

Conclusiones:

- $\uparrow$ Perspectiva de calidad para usuario
- $\uparrow$ Priorización de dificultad y utilidad
- $\uparrow$ Consenso sobre riesgos y no-riesgos
- $\uparrow$ Base documentada del modelo
- $\downarrow$ No evaluo costo
- $\downarrow$ No variaciones de escenarios (discreto)
- $\downarrow$ No es cuantitativo

Variaciones de escenarios: Comparar distintos posibles valores de respuesta (no binario). Curva utilidad/respuesta. Expresar concretamente QA.

6. Administración de requerimientos y SQA

Areas de conocimiento:

- Ingeniería de requerimientos
  - Development
    - Elicitation
    - Análisis
    - Especificación
    - Validación
  - Managment

Deseable de los reqs: Completo, correcto, factible, verificable, necesario, priorizado, no ambigüo.

Managment: Procesos relacionados con requerimientos: Cambios, agregados, asignación a iteraciones. Managment de cambios sobre artefactos $\in$ linea de base: Pasos de un SH desde un change request a un artefacto (en particular requirement change) hasta una decisión.

Tipos de cambio a requerimientos:

- Cambio: En contra o mayor a requerimiento aprobado
- Mejoras: "Peque nas" no expresadas en los requerimientos (= implican esfuerzo)

Gestión de cambio: Es importante el workflow. Objetivo: Controlar cambios (no evitarlos!). Las metodologías ágiles también tienen!

CCB (configuration control board). Equipo que toma decisiones de cambios. Puede haber varios según el tipo de cambio e incluso una jerarquía.

Desarrollo iterativo: El cambio es bienvenido! Al prinicipio hay muchos CU nuevos pero al final hay mayormente cambios a los existentes. $\Rightarrow$ ¿Como especifico cambios? ¿Seguimiento? ¿Alcance? ΔCU: Especifico solo la diferencia.

SQA: Conjunto de tareas que:

- Evalúan objetivamente la ejecución de procesos y los entregables
- Identifican, documentan y aseguran el correcto trato de desvios
- Provee feedback sobre QA

Objetivos de QA: (no garantiza que todo ande bien!)

- Dar visibilidad a la gerencia (especialmente desvios)
- Asegurar cumplimiento del proceso
- Ayudar a "poner la calidad" (a través de revisiones)