Resumen primer parcial

Ingeniería de Software

Busca dar un enfoque centralizado para desarrollar software.

- **Software:** es todo el conocimiento que se obtiene como resultado de un proyecto al aplicar un determinado proceso. Es un set de programas y la documentación que lo acompaña. Básicamente es conocimiento empaquetado. Es un producto de resultado único cada vez, siempre es diferente y por ello no se puede producir masivamente. Al no ser un producto físico, nunca se desgasta en el tiempo. Sin embargo, el software está constantemente sufriendo cambios, por ende preparamos nuestro software para que se pueda adaptar a dichos cambios. El software trabaja en 3 niveles:
    - **Proceso:** acá encontramos al ingeniero de proceso que definen el mejor proceso para llevar a cabo el desarrollo de un proyecto. (Definición mejor al final)
    - **Proyecto:** Se utilizan como unidad de gestión para el software que afecta recursos. Cada proyecto define una fecha de inicio y fin. Dentro del proyecto hay **personas** que ocupan diferentes roles. Por ejemplo: Líder de Proyecto. Las personas son la parte más importante del software y muy difícil de manejar.
    - **Producto de Software**

Problemas con el Desarrollo de Software:

Estos problemas casi nunca tienen que ver con la tecnología, sino que son causados, por lo general, por las personas.

- La versión final del producto no satisface las necesidades del cliente.
- No está documentado.
- No alcanza la calidad acordada.
- Se subestimó la complejidad. Invertís más tiempo y costo que lo esperado.
- Falta de comunicación

Cuando nos va bien es por:

- Involucramiento del cliente.
- Requerimientos bien definidos, pero aceptar que no tenemos el 100% de los requerimientos al comienzo del desarrollo y a su vez, que éstos pueden cambiar a medida que avanzamos.
- Personal competente
- Buena planificación
- Espectativas alcanzables o posibles

Gestión de Configuración de Software

Valen Opaganga Style: Es una disciplina de tipo “paraguas”, transversal a todo el proyecto con aplicación en las diferentes disciplinas. Su objetivo es mantener la integridad del producto de software. Busca evitar que se produzcan inconsistencias en el desarrollo de software (saber quién fue el último que modificó, las versiones, hacer que la propagación de un cambio sea consistente en todos los elementos de software, etc.).

¿Por qué lo usamos?

(Dice lo mismo que arriba)

Involucra para la configuración:

- Identificarla en un momento dado.
- Controlar sistemáticamente sus cambios: ver quién modificó, y cuándo.
- Mantener su integridad.

Problemas en el manejo de componentes

• Pérdida de un componente
• Pérdida de cambios (el componente que tengo no es el último)
• Sincronía fuente - objeto – ejecutable
• Regresión de fallas: con un click poder volver para atrás.
• Doble mantenimiento: no se puede tener cereza de que dos personas no estén trabajando sobre un mismo ítem al mismo tiempo. Dos personas podrían estar trabajando en la misma funcionalidad por ejemplo, sin saberlo. O sea se gastan recursos al pedo.
• Superposición de cambios: cambio algo, y otro superpone mi cambio con otro cambio.
• Cambios no validados: desarrollamos requerimientos innecesarios. No sé.

Integridad del Producto

Un Producto tiene Integridad cuando:

- Satisface las necesidades del usuario.
- Puede ser fácil y completamente rastreado durante su ciclo de vida: es decir poder obtener una visión entre las relaciones que tienen los diferentes artefactos, y la trazabilidad de los cambios en cada artefacto. (Es fácil ver cómo un cambio en un artefacto te afecta la arquitectura, etc.)
- Satisface criterios de performance.
- Cumple con sus expectativas de costo.

Ítem de Configuración: Todos aquellos artefactos que forman parte del producto o participan de la gestión del proyecto que pueden sufrir cambios o necesitan ser compartidos entre los miembros del equipo y sobre los cuales necesitamos conocer su estado y evolución.

Elementos o Actividades de la SCM

1) **Identificación de ítems de configuración:** busca determinar cuáles de todos los objetos, documentos, etc, pueden ser o son candidatos a ser ítems de configuración (por ejemplo: vista arquitectónica, casos de uso, código fuente, documentación de una clase, script para generar la BD, plan de proyecto, el contrato con el cliente, etc). Se identifican dichos ítems para:
    - Llevar control.
    - Porque cuando cambia el producto éstos también van a cambiar.
    - Necesitan ser compartidos por los miembros del equipo.
    - Tomar decisiones.

    Al principio se decide qué artefactos o documentos van a pertenecer a los ítems de configuración. Todo esto queda documentado en el "Plan de Gestión de Configuración", que también es un ítem de configuración.
    En esta etapa también se definen los repositorios.

    Una vez identificados los ítems de configuración, los ponemos en repositorios.
    - Repositorio: es un depósito que contiene ítems de configuración, mantiene la historia de cada ítem, sus atributos y relaciones. Es utilizado para evaluar el impacto de los cambios propuestos en los ítems de configuración. Hay distintos tipos de repositorios:
        - **Centralizados:** hay un repositorio padre al que acceden los clientes. La idea es almacenar ítems y obtener ítems. Los administradores tienen mayor control sobre los repositorios. Falla el servidor y estamos al horno.
        - **Descentralizados:** cada cliente va a tener una copia exactamente igual del repositorio completo, si un servidor muere, es cuestión de copiar y pegar. Estos repositorios son reemplazables por el repositorio del servidor. Ej: Github

2) **Control de Cambios:** tiene como objetivo principal definir un procedimiento que dé soporte a los cambios en el producto. Busca establecer un procedimiento formal, que ante un requerimiento de cambio, un conjunto de personas (comité de control de cambios) evalúan el impacto de un cambio y deciden si llevarlo a cabo o no. Esto es lo mismo que el párrafo de abajo

El Control de Cambios tiene su origen en un Requerimiento de Cambio a uno o varios ítems de configuración que se encuentran en una línea base. Es un Procedimiento formal que involucra diferentes actores y una evaluación del impacto del cambio

El Comité de Control de Cambios va a estar formado por representantes de cada área que pueda ser afectada por el cambio en el software.

3) **Auditoría de Configuraciones:** es una revisión objetiva e independiente, es decir que la persona encargada de realizar la auditoría, no debe formar parte del equipo de trabajo del proyecto.

   Para una auditoría se necesita una línea base marcada en el repositorio y el plan (que define cuándo se hace la auditoría). Reconocemos dos tipos de auditoría:

   - **Auditoría de Configuración Física:** se realiza antes que la funcional y sirve para verificar que un ítem es íntegro con su documentación asociada. Asegura que lo que está indicado para cada Ítem de Configuración en la línea base o en la actualización se ha alcanzado realmente.

   - **Auditoría Funcional de Configuración:** permite validar que los ítems de configuración se corresponden con los requerimientos del cliente.

   Características:

   - Cuestan tiempo y dinero.
   - Se deben realizar desde el principio a fin del proyecto. Su postergación a etapas posteriores puede llegar a hacer fracasar el proyecto.
   - Se define cuando se hace el plan del proyecto.
   - Suministra visibilidad y rastreabilidad del ciclo de vida del producto de software.

4) **Generación de Reportes e Informes de Estado:** busca obtener información de un repositorio y generar reportes que van a ayudar a la toma de decisiones. Se van a generar reportes de los impactos que pueda generar un cambio y además reportes de la historia de la evolución de un proyecto. Por ejemplo va a decir cuándo se modificó, quién lo hizo, etc.
   Incluye reportes de rastreabilidad de todos los cambios realizados a las líneas base durante el ciclo de vida.
   Sirve también para caracterizar cómo va evolucionando el sw con el paso del tiempo.

¿Qué es una Configuración?

Es una foto del repositorio en un momento dado con todos los ítems de Configuración asociados a una versión. Es decir, se resume en un conjunto de Ítems de Configuración con su estado, en un momento dado.

Línea Base: es una configuración que ha sido revisada formalmente. Necesitan un procedimiento formal de Control de Cambios para poder modificar los ítems que pertenezcan a la Línea Base.
Están condiciones de ser considerado como un punto de referencia (como un checkpoint).

Antes de que los Ítems pertenezcan a una Línea Base, podemos hacer cambios libremente, de manera informal.

En el Plan de Gestión de Configuración se establece cuándo se van a generar una Línea Base. No respecto al tiempo, sino respecto a las etapas, como por ejemplo cuando hay una reunión con el cliente y validamos algo, después de las pruebas o al finalizar un workflow.

Se usan etiquetas para identificar cada Línea Base.

Encontramos dos tipos de Linea Base:

- **Especificación:** son las que se hacen antes de tener el código.
- **Operacional:** son las que se hacen cuando ya tengo el código.

El Comité de Control de Cambios es el dueño de la Línea Base. Está a cargo de llevar el control, realizar modificaciones, etc.

Ramas

Son bifurcaciones del desarrollo principal. Facilitan el aislamiento de los ítems de Configuración. Se elige un conjunto de Ítems de Configuración y se genera una rama separada de la principal, sobre esta rama secundaria se realizan los cambios deseados y luego se integran.

La **integración de ramas** es el proceso mediante el cual se integran los cambios realizados en las ramas secundarias, en la rama principal. Esto se denomina Merge. Si dos personas quieren realizar cambios en un mismo Ítem de Configuración surge un conflicto, esto se puede resolver con diff aunque no siempre se puede.

Plan de Gestión de Configuración

La Gestión de Configuración también se planifica. Dicho plan tiene:

• **Reglas de nombrado de los CI:** una vez que los identifico, defino cómo se van a llamar y dónde se van a ubicar.

• **Herramientas a utilizar para SCM**
• Roles e integrantes del Comité: quiénes van a formar parte.
• Procedimiento formal de cambios: cuál va a ser el documento de gestión de cambios. Cuáles van a ver las actividades formales para, ante un requerimiento de cambio, evaluar su impacto, y ver si se realiza o no (y cómo se realizaría).
• Plantillas de formularios
• Procesos de Auditoría

SCM en Entornos Ágiles

- **Individuos e interacciones por sobre procesos y herramientas.** Los procesos y herramientas de SCM deben adaptarse al equipo y su forma de trabajo y no al revés (busquemos nosotros algo que nos sirva, antes que "el proyecto dice que usemos esto").
- **Software funcionando por sobre documentación extensiva:** voy a privilegiar lo que el cliente quiere, el sistema funcionando, antes que la documentación.
- **Colaboración con el cliente por sobre negociación contractual:** Tener una relación más colaborativa con el cliente y no tan profesional. Tocarle la gamba, no sé.
- **Respuesta ante el cambio por sobre seguir un plan** facilitar y propiciar el cambio en vez de prevenirlo. El sw cambia y hay que aceptarlo.

Tipos de Pruebas

1) **Smoke test**: Corrida genérica, antes del ciclo 0, para ver si la versión está en condiciones de ser testeada. Básicamente para ver que el sistema sea usable.
2) **Testing Funcional**: Se basa en pruebas de requerimientos funcionales de manera individual y luego en procesos de negocio. Pruebas basadas en funciones y características del sistema.
3) **Testing no funcional**: Se basa en pruebas de requerimientos no funcionales
    a) Pruebas de performance: Tiempo de respuesta y concurrencia (muy dependiente de los recursos)
    b) Pruebas de carga: Prueba el sistema en su capacidad máxima de recursos utilizados.
    c) Pruebas de stress: Pruebas al sistema por encima de su capacidad máxima. Se espera que el sistema falle y se busca analizar cómo se recupera (cuando dije que el sistema era para 50 usuarios y entran 100, cuánto tardaría en recuperarse?).
    d) Pruebas de usabilidad: Vínculo entre el usuario y el sw. Por ejemplo, cuántos clicks tiene que hacer el usuario para cumplir una función.
    e) Pruebas de mantenimiento: Prueba sobre el esfuerzo requerido para introducir cambios.
    f) Pruebas de fiabilidad: Qué tan seguro y confiable es mi sistema ante usos indebidos o fallas inesperadas en el uso habitual.
    g) Pruebas de portabilidad: Ver su funcionamiento en distintas plataformas y ambientes de Hardware.
4) **Pruebas de Configuración**: Ver si nuestro software funciona con APIS o aplicaciones ajenas íntegramente.
5) **Pruebas de Documentación**: Que los manuales de usuarios sean consistentes con la versión actual del sistema.

El testing y el ciclo de vida

Modelo en V

Promueve el hacer testing temprano. Propone el desarrollo de software de manera general a lo particular, y luego el testing de lo particular a lo general.

TDD: Test-Driven Development

Metodología de desarrollo que busca fusionar la programación con el testing unitario.

Primero programo el componente para probar el componente a desarrollar. Las pruebas son automáticas y van mejorando la calidad del código.

La calidad está en la simpleza: un código limpio (clean) es aquel que cumple su objetivo con el código más sencillo posible.

Es una técnica avanzada que involucra otras dos prácticas: Escribir las pruebas primero (Test First Development) y Refactorización (Refactoring). Este último consiste en, una vez que el programa pasa el testeo, reescribir el código buscando hacerlo lo más sencillo posible; luego se vuelve a testear y así sucesivamente.

Testing en metodologías ágiles

El testing en sí se hace igual sin importar la metodología de desarrollo. Lo que cambia es el “Cómo”, no el “Qué”.

Filosofía: “Mientras más automaticemos, mejor”.

El manifiesto del testing especifica la concepción que las metodologías ágiles tienen sobre el testing. Se pueden hacer todos los tipos de pruebas pero se tienen que respetar las siguientes consignas:

1) Se hacen a lo largo de todo el proceso en vez de hacerlo al final
2) Se trabaja para prevenir errores en vez de encontrar defectos (se fomenta el desarrollo de pares)
3) Se trabaja en la comprensión del código (o entendimiento del sistema) en vez de ver si funciona
4) Se trabaja en entregar el mejor software posible al cliente en vez de buscar romper el sistema.
5) La calidad es responsabilidad de todo el equipo, y no sólo del tester. El testing no incorpora calidad (solo detecta defectos). La calidad debería estar en el desarrollo.

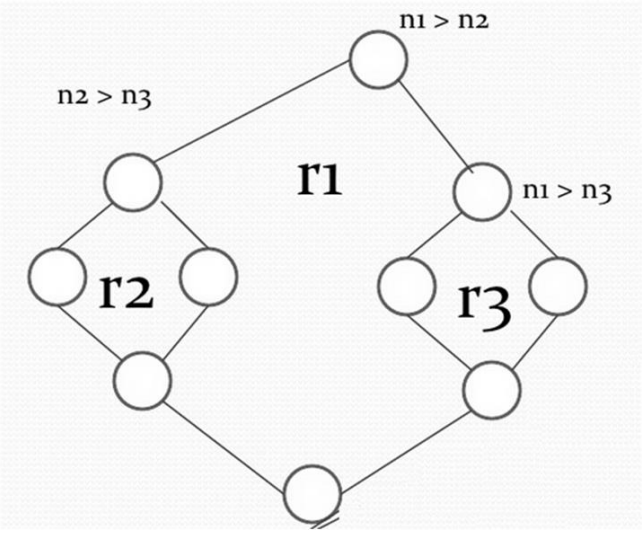
Hay 9 principios pero es un viaje estudiarlos. El más importante es “Las pruebas son partes del ‘done’”: Significa que el proceso de testeo es parte indispensable para definir que un producto está “terminado”, por ende no se debe omitir.

Testing de Caja Blanca

Cobertura de enunciados o caminos básicos

Permite tener una noción de cuán complejo puede ser un programa. Identifica los distintos caminos por los que puede recorrer una ejecución de un código. Se requiere poder representarlo en un grafo. El valor de la complejidad ciclomática (métrica que provee una medición de la complejidad lógica de un programa) se puede calcular mirando el grafo (cantidad de regiones +1) o utilizando la fórmula M = E − N + 2*P, donde:

M = Complejidad ciclomática.
E = Número de aristas del grafo
N = Número de nodos del grafo
P = Número de componentes conexos, nodos de salida



Cobertura de sentencias

Busca pasar por cada sentencia de una porción de código una vez.

Cobertura de decisión

Busca valuar todas las decisiones por verdadero y falso una vez. Cada if es una decisión, por más que tengan más de una condición adentro.

Cobertura de condición

Busca valuar cada condición por su valor verdadero y falso independientemente del valor de la decisión final del if.

Cobertura de decisión/condición

Ídem al anterior pero sí nos interesa el resultado de la decisión.

Cobertura múltiple

Combinatorio de todos los valores posibles en las condiciones de las decisiones

Revisiones Técnicas

También conocido como revisión de pares o entre colegas, es un instrumento de testing cuyo propósito es la detección temprana de errores.
Los errores en el software suelen surgir de un problema de comunicación entre los miembros de un equipo y los interesados.
Sirven para la validación y la verificación del sistema, esto es, validar que el sistema responda a los requerimientos especificados, y verificar que los cumpla sin errores.
Las revisiones técnicas las realiza un colega, no un grupo de testing o auditores, o alguien que tenga un vínculo jerárquico; es un par. Se busca detectar errores lo más temprano posible porque es más barato corregirlos antes de que se conviertan en defectos. El 80% del costo del desarrollo de sw está en el esfuerzo del personal.

Las revisiones técnicas no son gratuitas. Como tiene un costo, se debe planificar determinando el momento y el % de sw a revisar.
Detectar errores tarde también genera costos intangibles, como demoras en la entrega, y por ende, enojo del cliente.
Hay 3 tipos de fallos:

- Fallos cosméticos: no producen errores en el sw.
- Fallos menores: producen errores leves en el modelado del sistema
- Fallos mayores: generan errores graves que, en el código, producen que el sistema no funcione

De todas maneras siempre es relativo. Todo depende del impacto en el sistema, por ejemplo, un sw no comentado puede funcionar pero puede hacer que mi código sea inentendible e imposibilite el trabajo en equipo.
Las revisiones técnicas buscan trabajar por **prevención** (mientras que el testing busca **corregir** defectos), es decir, evitar convertir errores en defectos, y busca aseguramiento de calidad en el producto (el 50% del costo del sw es de retrabajo). En las metodologías ágiles se considera parte del "Done".
Las revisiones técnicas son siempre estáticas. No ejecutan el programa, sino que se revisa si el código está bien.

Métodos de Revisiones Técnicas

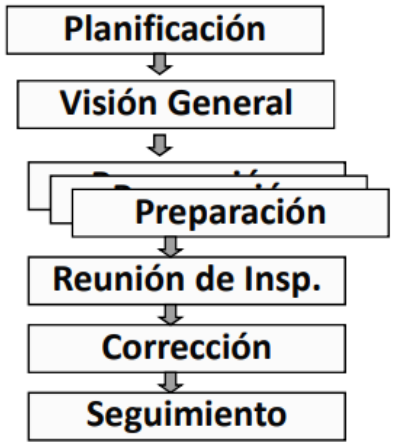
- Walkthrough: Son reuniones informales, en la que un autor le explica a otro cómo funciona el programa, y entre los dos detectan errores. No hay métricas, control ni formalidades.
  Técnica de análisis estático en la que un diseñador o programador dirige miembros del equipo de desarrollo y otras partes interesadas a través de un producto de software y los participantes formulan preguntas y realizan comentarios acerca de posibles errores, violación de estándares de desarrollo y otros problemas.
- Inspección: es un proceso formal, lo cual implica que existen checklists, métricas y controles. Su objetivo es detectar y remover la mayor cantidad de defectos de manera eficiente y eficaz.
  Es un proceso que busca detectar fallas, y no busca soluciones.

Se definen una serie de roles (tener en cuenta que hay roles que pueden ser interpretados por 1 misma persona), los cuales deben cumplirse siempre:

Rol	Responsabilidad
Autor	•Creador o encargado de mantener el producto que va a ser inspeccionado. •Inicia el proceso asignando un moderador y designa junto al moderador el resto de los roles •Entrega el producto a ser inspeccionado al moderador. •Reporta el tiempo de retrabajo y el nro. total de defectos al moderador.
Moderador	•Planifica y lidera la revisión. •Trabaja junto al autor para seleccionar el resto de los roles. •Entrega el producto a inspeccionar a los inspectores con tiempo (48hs) antes de la reunión. •Coordina la reunión asegurándose que no hay conductas inapropiadas • Hacer seguimiento de los defectos reportados.
Lector	Lee el producto a ser inspeccionado.
Anotador	Registra los hallazgos de la revisión
Inspector	Examina el producto antes de la reunión para encontrar defectos. Registra sus tiempos de preparación.

El anotador es el que lleva el tiempo de la reunión, el cual no debería superar las 2 horas, porque si se extiende se dejan de encontrar errores.
El autor es el responsable de corregir los errores una vez finalizada la reunión.

Actividades de la Inspección:



- **Elegir equipo, preparar material y calendario**
- **Presentar proceso y producto**
- **Análisis individual para encontrar potenciales defectos**
- **Análisis del equipo para recolectar potenciales defectos previos, filtrar falsos positivos**
- **Corregir defectos**
- **Verificar correcciones, recolectar datos**

Procesos Definidos vs. Empíricos

¿Qué es un proyecto?¿Qué es un proceso?¿Cómo se relacionan?

Un **proceso** es un conjunto de actividades estructuradas que utiliza recursos para lograr un objetivo. Es un método, un conjunto de lineamientos, una forma de hacer algo, de manera genérica y que abarca la mayor cantidad de escenarios posibles.
Un **proyecto** es una unidad de gestión y organización que administra recursos para lograr un objetivo. Es el medio que utilizo para obtener un producto de software. Un proyecto se basa en un proceso el cual le define qué hacer, pero no necesariamente hace todo lo que plantea. Un proyecto es único.

Procesos Definidos

Es un proceso ya existente y escrito, el cual abarca todas las alternativas posibles, y de manera organizada plantea una estructura de actividades a realizar para lograr un objetivo. Un proceso definido tiene las siguientes características:

- Resultados únicos: cada software es una pieza diferente.
- Fecha de inicio y fin: en el escenario feliz, la fecha fin depende del objetivo, y es cuando se cumple.
- Tiene un objetivo claro (conciso) y alcanzable (que se pueda cumplir).
- Elaboración gradual: el proyecto se lleva a cabo de a pasos, y un proceso ayuda.

Ciclo de Vida o Modelo de Proceso

Ciclo de vida del producto: todos los estados por los que pasa desde la idea de concebirlo hasta que se descarta.
Ciclo de vida del proyecto o modelo de proceso: dice cómo van a evolucionar las cosas dentro del **proyecto**. Puede abarcar uno o varios productos.
El ciclo de vida del producto es mucho más largo que del proyecto, porque cada iteración de un producto es un proyecto diferente.

Proceso → qué hacer
Proyecto → cómo hacer (el orden y cuándo hacerlo)

Modelos de proceso

- Cascada
  - Clásico
  - Con retroalimentación
  - Con subproyectos
- Iterativo → Manifiesto Ágil
- Incremental
- Iterativo e Incremental
- Espiral

Existen 3 tipos de ciclo de vida:

1. Secuenciales: como el cascada.
2. Iterativos: divide el proyecto en partes y va entregando. Los iterativos e incrementales van entregando de a partes, pero integradas cada vez haciendo algo con más funcionalidades.
3. Recursivo: como el espiral, muchos controles de riesgo.

Plan de Proyecto basado en Procesos Definidos

1) Elegir proceso y ciclo de vida (o modelo de proceso)
2) Definir objetivos y alcances del proyecto: el alcance del proyecto es todo el trabajo y sólo el trabajo a hacer para cumplir el objetivo (por ejemplo: desarrollar un producto de software que permita…)
3) Estimaciones: no hay certezas, factores de probabilidad. Se estiman (y en este orden):
   a) Tamaño: por casos de uso o funcionalidad, no por líneas de código.
   b) Esfuerzo: horas persona lineales: se asume que hay una persona haciendo una cosa por vez. El estimado del esfuerzo es subjetivo.
   c) Tiempo: 1 hora por día para algo que reclama 100 horas llevaría 100 días → no es lineal. La programación del proyecto se basa en esta estimación para “volcar la bolsa de horas disponibles en un cronograma”.
   d) Costo: depende de todo lo anterior. De base el 80% de los costos se basan en el esfuerzo humano (el resto suele distribuirse en costos administrativos, conectividad, etc.).
4) Definir recursos necesarios para el proyecto.
5) Hacer análisis de riesgos: un riesgo tiene asociado un factor de probabilidad. Si ocurre, puede afectar el desarrollo del proyecto. Con los riesgos se puede:
   a) Gestión proactiva: reconocer la existencia de los mismos e invertir en evitarlos o reducir su probabilidad de ocurrencia (es costoso).
   b) Gestión reactiva: ignorarlos. Cuando suceden ahí se hace algo al respecto.
6) Definir controles: de cómo se va cumpliendo el plan para hacer ajustes. (Por ejemplo: ¿cada cuánto nos juntamos con el cliente? ¿con el equipo?)

7) Definir métricas: que ayudan al control. Se define con qué herramientas se van a tomar mediciones, con qué criterios, etc.
Hay métricas para medir el producto (cantidad de errores por ejemplo), el proyecto (cantidad de iteraciones realizadas) y el proceso (eficiencia en la remosión de defectos: la calidad del producto depende de qué tan controlado está el proceso).

Procesos Empíricos

Surge como un intento de equilibrio entre no aplicar procesos y aplicar procesos muy estrictos y largos. Se basa en la experiencia del equipo de desarrollo, aprendiendo de errores de prácticas anteriores, y como cada proyecto es único, no quiere que un proceso defina estrictamente qué hay que hacer de antemano.
El equipo del proyecto configura el proceso que va a utilizar, pero el ciclo de vida será siempre el iterativo (en el caso del agilismo).
Como se basa en la experiencia, necesito retroalimentación constante. Surgen ciclos de “inspección-adaptación”.
El empirismo se enfoca en hacer lo que es de valor para el cliente, en vez de tanta documentación. De todas maneras se sigue documentando, modelado, diseño…
Los equipos de trabajo están autoorganizados: son todos pares, colegas, y entre todos toman decisiones; y están autogestionados: cada uno sabe qué tiene que hacer. Se basan en la premisa de que un equipo de trabajo motivado, capacitado, colaborativo, responsable y multidisciplinario (todos pueden hacer todo); además deben estar sincronizados para que no hayan 2 haciendo lo mismo innecesariamente.