Ingeniería en Software

Resumen Segundo Parcial

UNIDAD 5.1 - SOFTWARE CONFIGURATION MANAGEMENT

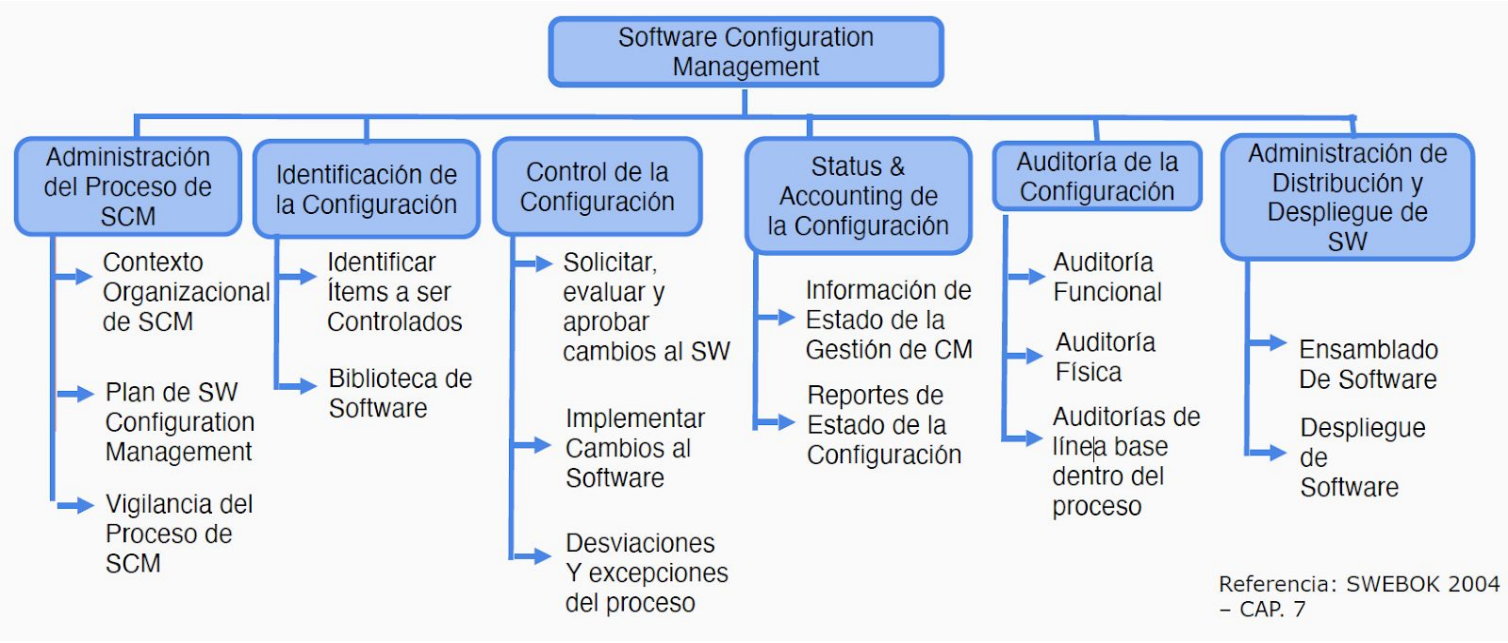
Gestionando cambios al software

SCM

Disciplina orientada a **administrar la evolución** de productos, procesos y ambientes.
Propósito: mantener la **integridad** de los productos del proyecto software.

Integridad: “que todo esté en armonía de forma consistente para que lo que se entregue sea lo que se efectivamente querés entregar”.

Todo esto está en en SWEBOK:



Beneficios

- Facilita el desarrollo y las actividades de implementación de cambios

ÍTEMS DE CONFIGURACIÓN

SWEBOK: *A configuration item is an item or aggregation of hardware or software or both that is designed to be managed as a single entity. Selecting SCIs is an important process in which a balance must be achieved between providing adequate visibility for project control purposes and providing a manageable number of controlled items.*

Cualquier elemento involucrado en el desarrollo del producto, en un momento en el tiempo:

- Código
- Servers
- Especificaciones de usuario
- Scripts
- Modelos
- Bases de datos

No todo necesita estar bajo SCM.
Para cada ítem se conoce:

- Nombre, Versión, Fecha de creación, Autor

CONFIGURACIÓN

SWEBOK: *Software configuration is the functional and physical characteristics of hardware or software as set forth in technical documentation or achieved in a product. It can be viewed as part of an overall system configuration*.

Todo el conjunto de ítems de configuración que son compilados en un ejecutable; todo lo que nos interesa mantener unido junto al entregable de cada proyecto. **Definen una "versión"**.

Una configuración no necesariamente es un release: hay que tener los cambios que se le hacen al aplicativo a lo largo del tiempo.

- **Identificarla** en un momento dado
- **Controlar sus cambios** sistemáticamente
- Mantener su **integridad** y origen

0. ADMINISTRACIÓN DEL PROCESO SCM

Etapa previa al arranque. Cuál es mi punto de partida a la hora de arrancar con SCM, cuál es el contexto organizacional.

- ¿Existe algún proceso de SCM en la organización? ¿Qué establece?
- ¿Qué herramientas hay disponibles?
- ¿Qué cosas deberíamos incluir o cambiar en el plan de SCM?

Actividades (SWEBOK)

- **Contexto Organizacional** para SCM y las relaciones entre los elementos de la organización
- **Restricciones y guía** para el proceso de SCM
    a. Por políticas/procedimientos organizacionales
- **Planear** el proceso SCM
    a. Actividades cubiertas (generalmente) por el plan:
        i. Software Configuration Identification (SCI)
        ii. Software Configuration Control (SCC)
        iii. Software Configuration Status Accounting (SCSA)
        iv. Software Release Management and Delivery (SRMD)
    b. Otras cosas incluídas:
        i. Organización, responsabilidades (roles)
        ii. Recursos y tiempos
        iii. Selección de herramientas
        iv. Vendor/Subcontractor Control
        v. Control de interfaces (entre componentes)
        vi. Estrategias de branching & merging.
    c. Se obtiene el documento "**SCM Plan**".
- **Vigilar** el SCM
    a. Para asegurarse que el SCM Plan se cumpla

1. IDENTIFICACIÓN DE LA CONFIGURACIÓN

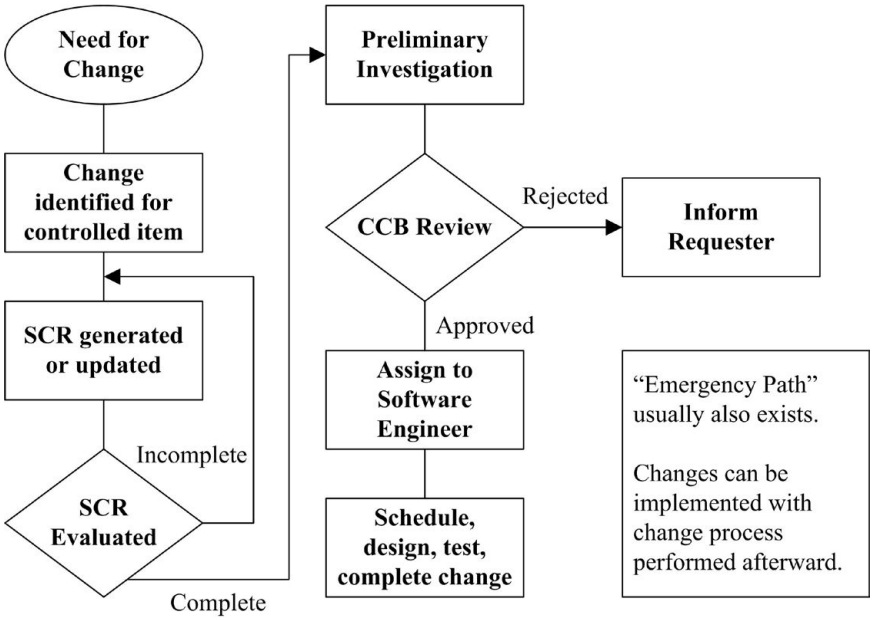
- Definir **qué ICs** están controlados/alcanzados por la gestión de configuración.
    - Identifica los **ICs**
    - Identifica las **relaciones entre los ICs**
        - Una **Versión** indica un estado del ítem
    - Define momentos (o condiciones) para
        - establecer un **baseline**
            - Versión aprobada de la configuración en un momento dado
        - o hacer un **release**
            - Distribución del software para fuera de la actividad de desarrollo

2. CONTROL DE LA CONFIGURACIÓN

Teniendo la configuración identificada, debemos establecer un orden para hacer los cambios en el software, para mantener la **integridad**.

- Identificación del **propósito** del cambio

- La evaluación del **impacto** y **aprobación** del cambio
    - **Software Configuration Control Board (SCCB)**: deciden si los cambios efectivamente tienen sentido y si vale la pena realizarlos.
- La **planificación de la incorporación** del cambio
- El **control** de la **implementación** del cambio y su verificación



3. STATUS & ACCOUNTING

Registrar y reportar la información necesaria para administrar la configuración de manera efectiva:

- Listar los **ICs aprobados**
- Mostrar el **estado de los cambios** que fueron aprobados
- Reportar la **trazabilidad** de todos los cambios efectuados al baseline
    - Construir la historia: "Lo probamos con esta configuración pero en producción falló. Veamos qué tengo mal en la configuración de test."
    - Autores

Debe contestar:

- Cuándo cambió, quién, qué, alcance, quién aprobó, quién solicitó.

Ejemplo: Github Insights.

4. ADMINISTRACIÓN DE DISTRIBUCIÓN Y DESPLIEGUE DE SW

Una vez que hicimos los cambios, debemos construir nuestra configuración y habilitarla para que pueda salir a producción:

Asegurar la **construcción exitosa (Software Building)** del paquete de software para luego **liberarlo** en forma controlada a otros entornos **(Release Management)**.

SWEBOK

- ***Software building*** *is the activity of combining the correct versions of software configuration items, using the appropriate configuration data, into an executable program for delivery to a customer or other recipient, such as the testing activity*.
- ***Software release management*** *encompasses the identification, packaging, and delivery of the elements of a product—for example, an executable program, documentation, release notes, and configuration data.*

Comprende el ejecutable y la administración, identificación y distribución de un producto.

5. AUDITORÍA DE LA CONFIGURACIÓN

SCM busca integridad de todo el proyecto. La auditoría saca un snapshot y dice “En este momento, ¿tiene un hueco de integridad?”.

Objetivo: Realizar una **verificación del estado de la configuración** a fin de determinar si se están cumpliendo los requerimientos especificados.

Auditoría funcional

SWEBOK: Verifica el cumplimiento de los **requerimientos**.

Una forma es testing. Otra es (x ej para el servicio que actualiza la contabilidad), hacer un diff y si las líneas de código cambian exclusivamente lo solicitado.

Auditoría física

SWEBOK: *Ensure that the* ***design*** *and* ***reference documentation*** *is consistent with the as-built software* ***product***

Verifica la configuración del producto en cuanto a la **estructura** especificada.

Va al detalle de la implementación; que **lo que dijimos que iba a cambiar efectivamente cambió**. X ejemplo “mis requerimientos tienen que estar en Jira cargados una manera determinada”. Que todos los requerimientos estén en Jira.

Auditoría de proceso

Que se haya cumplido el proceso de SCM en todo el proyecto. X ej que los cambios tienen que estar aprobados por el CCB. Lo aprobó Pablo y Alejo. ¿Son Alejo y Pablo el CCB para el cambio? No, entonces hay que bajarles el sueldo.

SWEBOK: *audits can be carried out during the development process to investigate the current status of specific elements of the configuration. In this case, an audit could be applied to sampled baseline items to ensure that performance is consistent with specifications or to ensure that evolving documentation continues to be consistent with the developing baseline item*

BUT I ONLY CHANGED ONE LINE OF CODE

Software Configuration Management

“El arte de identificar, organizar y controlar modificaciones a software”.
Aplicado durante todo el proceso de desarrollo y mantenimiento.
Determina un set bien especificado del conjunto de partes de un software y los procedimientos y herramientas exactos para construir el producto (y sus distintas versiones) con dichas partes.

Configuration Identification

“La parte más importante”
Identificar los Software Configuration Items (**SCIs**):
Partes necesarias para diseñar, desarrollar, construir, mantener, testear y deployar.
En general, es *cualquier* cosa que se necesite para realizar estos puntos.
Objetivo: lograr la integridad del proyecto.

- Ítems fundamentales para el proyecto
- Ítems que cambien en el tiempo
- Ítems que tengan relaciones entre sí
- Ítems que van a compartir entre distintas personas en el proyecto

Línea base: el grupo de SCIs que se toman como punto de regreso (safecheck). Implica que el equipo y el cliente den el ok.

Configuration Control

El proceso de **controlar y limitar** los cambios al software.
Asegurarse que los cambios a un SCI solo sucedan luego de su análisis, evaluación, revisión, y aprobación por un grupo que controla los cambios (Configuration Control Board).

El **CCB** se asegura de que los impactos de los cambios son correctamente evaluados. Deciden si vale la pena.
Lo forman: program management, systems engineering, software engineering, software quality assurance, software configuration management, independent test, y un customer representative.

Configuration Status Accounting

Status accounting is the SCM activity responsible for tracking and maintaining data relative to each of these SCI’s (trazabilidad de los cambios).
Se logea información para trazar:
What changes have been **requested**?
What changes have been **made**?
When did each change occur?
What was the **reason** for each change?
Who authorized each change?
Who performed each change?
What **SCIs were affected** by each change?

Configuration Auditing

Audits should be conducted to help assure that software development policies, processes, and procedures are being consistently followed and adhered to. Audits verify that the software product is built according to the requirements, standards, or contractual agreement.

UNIDAD 5.2 - EL PROCESO DE SCM: DEFINICIONES

SCM aplicado en la vida real

Problema SISOP: Se hace un merge y funciona pero rompe en producción (???)

PLAN DE SCM

- Lista **ICs**, y en qué momento ingresan al sistema de CM
    - Cómo se registran cambios al SW (Ej con Jira)
    - Se establecen roles que aprueban o rechazan al cambio (CCB)
    - Están identificados los repositorios de los artefactos
        - Github como repositorio de código
        - sonatype como repositorio de paquetes
        - npm como gestor de paquetes
- Define **estándares** de nombres, jerarquías de directorios, estándares de versionamiento
- Define políticas de **branching** y de **merging**
    - Están identificados los momentos en los que se realizan las baselines.
        - Ej: gitflow como flujo de trabajo
- Define los **procedimientos** para crear "**builds**" y "**releases**"
    - Los cambios son gestionados completamente, desde su ingreso hasta su puesta en producción
- Define las reglas de **uso de la herramienta de CM** y el rol del administrador de la configuración
- Define el **contenido** de los reportes de **auditoría** y los **momentos** en que estas se ejecutan

Puede ser definido por proyecto o a nivel organizacional (y luego realizar una adaptación al proyecto).

BASELINE

Un **estado** de la configuración de un conjunto de ICs en el ciclo de desarrollo, que puede ser tomado como **punto de referencia** para una siguiente etapa del ciclo.

Se establece porque se verifica que esa configuración satisface algunos requerimientos.

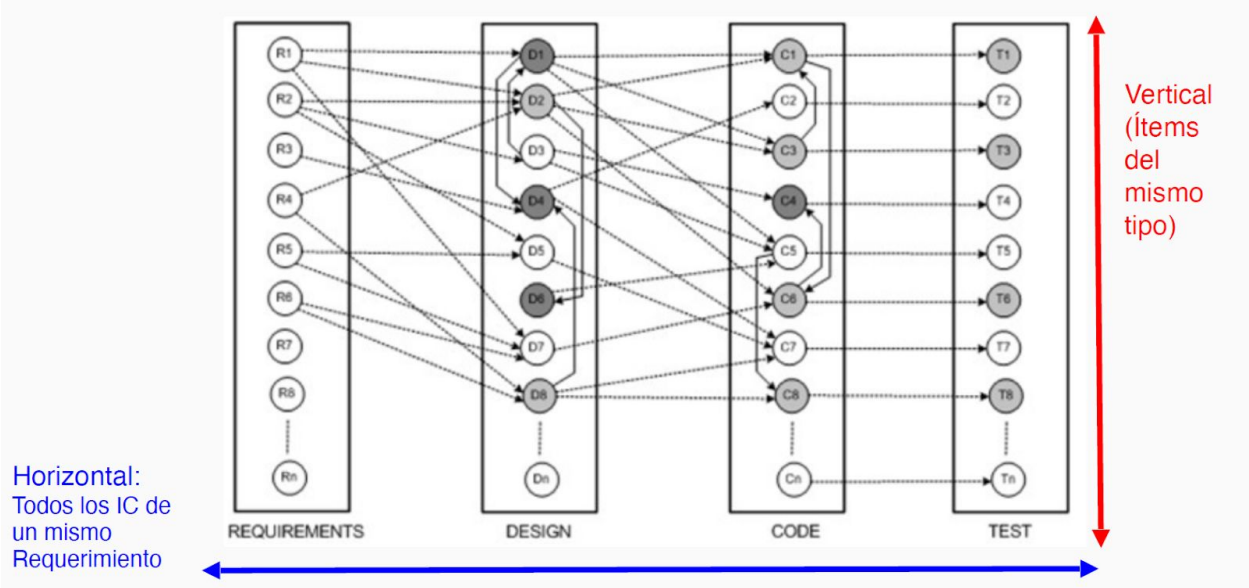
Para un baseline, hay un conjunto de cambios a los ICs con respecto al último baseline.

Un baseline y un release no son lo mismo. Cuando hay un release normalmente hay un baseline (puede pasar que no xq hay un fix urgente que se libera sin armar el baseline).

BRANCH

Línea de desarrollo separada. Utiliza un baseline del repo como punto de partida.
Permite trabajar en múltiples versiones de un producto utilizando el mismo set de ICs.

TRAZABILIDAD ENTRE ÍTEMS DE CONFIGURACIÓN



El CCB (Configuration Control Board) se fija acá qué hay que tocar si quiero cambiar x cosa.

UNIDAD 5.3 - RELEASE & DEPLOY

Distribución y despliegue del software

RELEASE MANAGEMENT

Asegurar la **construcción exitosa** del paquete de software (basada en los ICs requeridos para la funcionalidad a entregar) para luego **liberarlo** en forma controlada a otros entornos ya sea de pruebas, producción, usuario final, etc.

SOFTWARE BUILDING

Los builds:

- Deben ser **automáticos**
- Deben permitir la generación de **reportes** del estado del mismo

Estos dos puntos traen como beneficio:

- Reducen la **cantidad de defectos**
- Mejora la reproducción de problemas y **trazabilidad**
- Mejora la **performance del equipo** de desarrollo

Tipos de Builds

- **Local** builds: las hace el dev, **corre unit tests**
- **Integration** builds: para **generar el entorno** completo para pruebas de integración
- **Nightly** builds: builds diarios con **reportes de su estabilidad**, tiempo de build, etc.
- **Release** builds: Se disparan cuando un administrador lo decide, o por el mismo sistema de integración si se utiliza el modo de deployment contínuo

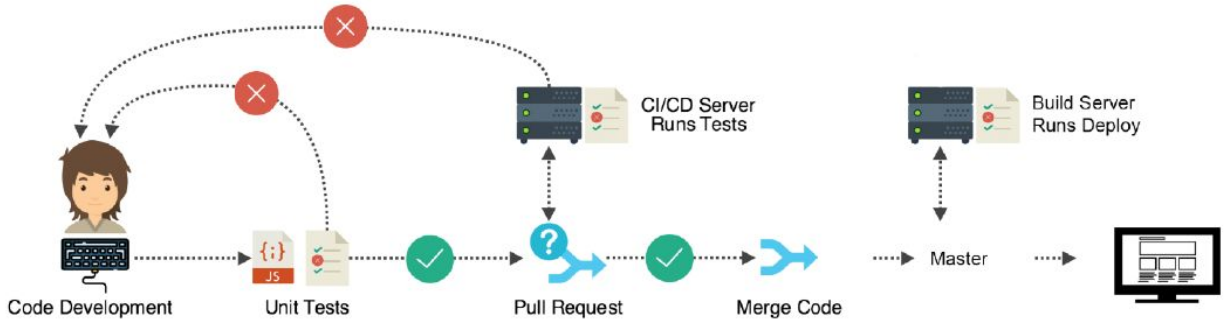
SOFTWARE DEPLOYMENT

Qué debería hacer una vez que tengo , por ejemplo el código corriendo el localhost.
Se busca el mecanismo más efectivo para hacer despliegue controlado a los distintos usuarios según:

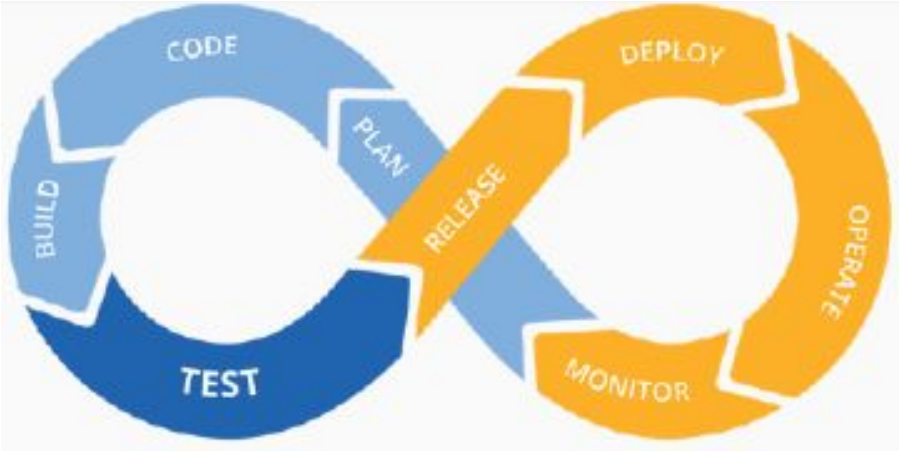
- Qué **usuarios** deberán recibir los cambios (geográfico, premium, por segmento)
- En qué forma se deberá hacer el despliegue
    - **Entornos** por los que deberá pasar (testing, staging, producción, otros)
    - Procesos de **Roll Forward**
    - Procesos de **Rollback**
    - **Validación** de despliegue correcto / incorrecto
- **Riesgos** del despliegue y cómo se minimizan
- **Aprobación** por el negocio y/o área de **QA**
- **Quienes** realizarán el deployment de la versión definida

BUILD & DEPLOYMENT PIPELINES

La manifestación **automatizada** del **proceso** para **llevar el software** desde la aprobación del cambio hasta manos de los usuarios.
Desde que se compila y construye el software, seguido del progreso a través de varias etapas de testing y deployment.



INTEGRACIÓN CONTINUA (CI) Y DESPLIEGUE CONTINUO (CD)



Integración Continua

Realizar **integración de código** al menos una vez por día para minimizar problemas de código.

- **Repositorio** de código **único**
- **Automatizar** el proceso de **build**
- Hacer el **build testeable automáticamente**
- Todo commit debe construirse por una herramienta de integración (no por el dev)
- El **build** debe ser **rápido**

Continuous Delivery (Entrega Continua)

Asegurar que el software puede desplegado en producción rápidamente y en cualquier momento. Hace todo el pipeline pero **a producción va manual**.

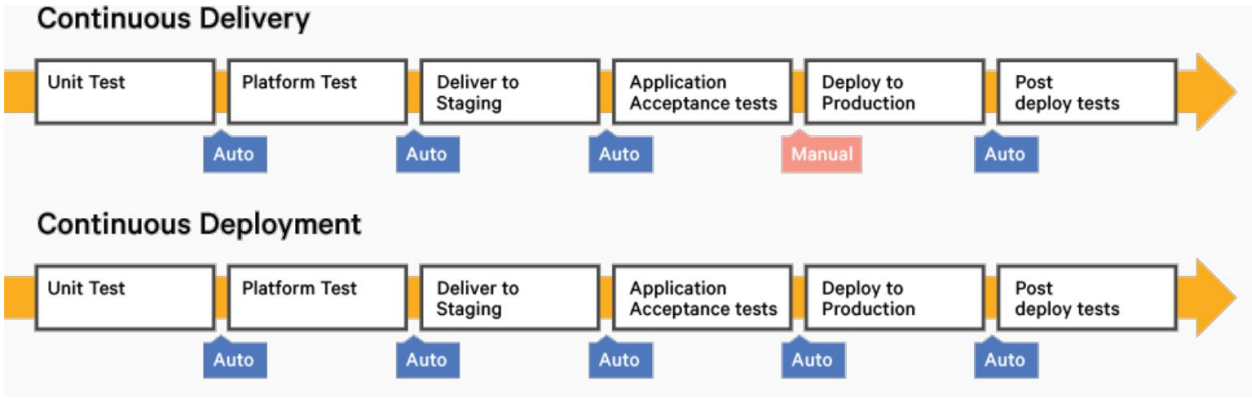
Lo logra haciendo que cada cambio llegue a un entorno de **staging**, donde se corren **pruebas**. Pasa a producción cuando alguien apruebe el cambio (**manualmente**).

Nota: **Continuous Deployment** *(Despliegue Continuo)* es lo mismo pero además va directo a producción: el despliegue se realiza en forma automática por un proceso, no por personas.

Pasos:

- Unit test
- Platform Test
- **Deliver to staging**
- Application acceptance tests (**manual en C Delivery, automático en C deployment**)
- Deploy to production
- Post deploy tests

CD y CI se complementan.



DEVOPS

DevOps trata de terminar con la cultura de separación y **falta de comunicación y colaboración** entre los devs y lo de Ops (release/ deploy/ operate/ monitor).
Permite definir la estructura describiendola, codeandola en las especificaciones (por ejemplo con Docker).

DevOps: conjunto de prácticas destinadas a reducir en tiempo entre: el **cambio** de un sistema y su **pasaje a producción**, garantizando calidad y minimizando el esfuerzo.

Prácticas

- Planificación del Cambio
    - Conversación y colaboración
- Coding & Building
    - Automated Build Pipelines
    - Infrastructure as Code
- Testing
    - Automated Testing
    - Chaos Testing / Fault Injection
- Release & Deployment
    - Continuous Integration
    - Continuous Delivery
- Operation & Monitoring
    - Virtualization & Containers
    - Monitoring & Alerting tools

CALMS

Una filosofía de trabajo

- **C**ultura: Ser dueños del cambio para mejorar la colaboración y comunicación.
- **A**utomatización: Eliminar el trabajo manual y repetitivo lleva a procesos repetibles y sistemas confiables, reduce error humano.
- **L**ean: Remover la burocracia para tener ciclos más cortos y menos desperdicio
- **M**étricas: Medir todo, usar datos para refinar los ciclos.
- **S**haring: Compartir experiencias de éxito y falla para que otros puedan aprender.

UNIDAD 6 - **TESTING**

CONCEPTOS GENERALES

Definiciones

- Según IEEE
    - **Actividad** en la cual un sistema es **ejecutado** bajo condiciones específicas
    - Los **resultados** son observados para realizar una **evaluación** de algún aspecto del sistema
- Definición usual
    - Actividad que compara los resultados obtenidos con los esperados
    - Propósito: determinar si el sistema se encuentra libre de defectos
- Definición según Kaner
    - Investigación técnica y empírica
    - Propósito: proveer información de la **calidad del producto a los stakeholders**
        - Calidad = cumple con los requerimientos. No tiene fallas al ser usado

Características

- Las pruebas encuentran fallas, NO defectos
    - Demuestran la presencia de defectos, no su ausencia
- Partimos de la premisa de que el SW a probar contiene defectos
    - El objetivo no es probar que el SW no los tiene sino encontrar tantos como sea posible!!

Importancia del Testing

- Los bugs pueden ser costosos y/o peligrosos
    - Potencialmente pueden causar problemas económicos o pérdidas humanas
- Cada vez hay SW más complejo y estricto

Crisis del Software

Principales inquietudes (de las cuales al aplicarles el método científico nació la Ingeniería del Software y la Ingeniería de Requerimientos):

- **Tiempos** de desarrollo largos
- **Costes** elevados y difíciles de calcular

- **Errores** difíciles de encontrar
- **Progreso** difícil de constatar

Objetivo del Testing

Encontrar las fallas del producto

- De forma eficiente
    - **rápido** y **barato**
- De forma eficaz
    - encontrar la **mayor cantidad** de fallas
        - las más **importantes**
    - no detectar fallas que en realidad no son (falsos positivos)

Causas de las fallas

- **Requerimientos**
    - poco claros
    - incorrectos
    - cambiantes
- **Diseño** y/o lógica complicada
- Desarrolladores y/o Testers sin **experiencia** o con poco conocimiento
- Falta de **tiempo**
- Problemas de **entorno**

ASEGURAMIENTO DE LA CALIDAD (QA)

- Asegurar que el producto **cumple con los requerimientos** técnicos establecidos
- Se usa un **patrón** planificado y sistemático
- Incluye las **Pruebas de Software**

Controlar la calidad (QC)

- La calidad del producto depende de las tareas realizadas durante el proceso, no al final
- Detectar errores de forma temprana
- **QC vs QA**:
    - Terminé de construir un auto y lo pruebo: **QC**. Tengo que tener todo armado y ahí probarlo.
    - Controlar el proceso de los pasos intermedios: **QA**

        - Busca asegurarse que el proceso lo estás ejecutando correctamente y no esperar a tener algo terminado para ver si tenés algo bien o mal.
        - Podría tener QC de los productos terminados de cada paso
    - Ej: si mi **proceso** dice que se tienen que dar 7 vueltas a la tuerca, QA controla que se dieron 7 vueltas por más que con 4 pase QC porque está bien colocada.
- El **costo de QC** es más barato que la falla en producción

Verificación vs validación

- Verificación
    - Construir el producto **correctamente**
    - Cómo sé si lo estoy construyendo correctamente:
        - Pruebas
            - **Unitarias**
            - **De integración**
            - De regresión
            - De sistema
- Validación
    - Construir el **producto correcto** (para el cliente)
    - Puede pasar las otras pruebas, pero no ser lo que quiere el cliente
    - Pruebas
        - **De aceptación de usuario (UAT o user acceptance testing)**
        - **De UI/UX**
        - De regresión
        - De sistema

PROCESO DE TESTING

Probar: ejecutar un componente con el objetivo de producir fallas. Si se produjeron, la prueba fue un éxito (!!! porque testeo para encontrar fallas) (y si no encuentro fallas es un fracaso??).

- Plan de prueba
    - **Requerimientos** (cualquier formato: User Story, un papel escrito, ...)
    - **Condiciones de prueba** en base a los requerimientos
    - **Casos puntuales de prueba** (ej: hacer un pedido de 5 hamburguesas en la sucursal de corrientes de mc donalds)
    - Tengo que tener un **resultado esperado**, por ej el precio
- Ejecución de prueba
    - **Componente**: un web service, una funcionalidad, ...
    - Armo un **entorno** para probarlo, que me alimenta los casos
    - Ejecuto por caso y veo los resultados
        - Si me aparece algún resultado distinto al esperado no es una falla sino un *incidente*, xq puede ser un falso positivo.
            - Por ejemplo estoy haciendo mal los pasos al crear un objeto en el test. No es una falla.

Conceptos relacionados al proceso

1. Equivocación

- Acción **humana** que produce un resultado incorrecto (*Ej: confundir > con <)*

2. Defecto

- **Proceso** incorrecto

- Ausencia de cierta **característica**
- Es la causa de la falla

3. Falla

- **Resultado** de la ejecución incorrecta (puede o no manifestarse)
- Es el efecto del defecto

Incidentes

- Ocurrencia de un evento durante la ejecución de una **prueba** que requiere investigación
- Se deben, entre otros, a defectos en
    - el software
    - los casos de prueba
    - el entorno
- No todos los incidentes son fallas (ej: defectos en los casos de prueba)

Depuración

- Proceso para **eliminar un defecto** que posee el software
- **No es una tarea de prueba**, sino consecuencia de ella
- La depuración puede introducir nuevos defectos

Proceso de depuración

1. **Detectar**
    - La **prueba** detecta la **falla** (efecto) de un defecto
    - La **depuración** detecta el **defecto** (causa)
2. **Depurar**
    - **Eliminar** el defecto
    - Encontrar la **razón** del defecto
    - Hallar una solución y aplicarla
3. Volver a **probar**
    - Asegurar que **sacamos el defecto** (y que no introducimos otros)
4. **Aprender** para el futuro

ECONOMíA DEL TESTING

Hasta cuándo tengo que probar

Premisas

- **Probar**: **establecer confianza** en que un programa hace lo supuesto
- No se puede demostrar que un programa es correcto
- Continuar probando es una decisión **económica**

Estrategias

Es imposible estimar la intensidad de testing para alcanzar una determinada densidad de defectos. Estrategias (cortar cuando...):

- El software supera mi **conjunto de pruebas** diseñado
- Se acepta cierta **cantidad de fallas** no críticas (“good enough”)
- La cantidad de fallas **detectadas** es similar a las **estimadas**

Abaratamiento

- Para abaratar y acelerar el testing, sin degradar su utilidad, se debe diseñar un software apto para ser testeado (**software testeable**)
- Técnicas
    - Diseño **modular**
    - Ocultamiento de información
    - Puntos de control (**breakpoints?**)
    - Programación **no egoísta**

ENFOQUES DE PRUEBA

Caja negra

- Prueba funcional, producida por datos o la entrada/salida
- Nos desentendemos del comportamiento y estructura interna del componente a probar

Condiciones y casos

- **Condiciones** de prueba
    - Descripciones de **situaciones** a las que el sistema debe responder

      (probar que el ticket solo funcione antes del vencimiento)

- **Casos** de prueba
    - **Lotes de datos** necesarios para que se cumpla una condición de prueba

      (probando con fecha 15/6 y vencimiento del ticket 14/6)

Criterios de selección de casos de prueba

- **Variaciones de eventos**
- **Clases de equivalencia**
    - Dividimos todos los casos de prueba en clases de equivalencia

Condiciones de Prueba

Edad	TC	Res. Esperado
< 21	Válida (Visa|MC)	X
< 21	Inválida	X
>= 21 y < 80	Válida (Visa|MC)	OK
>= 21 y < 80	Inválida	X
>= 80	Válida (Visa|MC)	X
>= 80	Inválida	X

        - **De entrada**
        - **De salida**
        - Cada fila del ejemplo sería una clase de equivalencia
    - Todos los casos de una clase son equivalentes entre sí
        - detectan los **mismos** errores
    - Seleccionamos algunos ejemplos de cada clase para cubrir todas las pruebas
    - El éxito está en la **selección de cada clase**:
        - 1. **Identificar** las clases de equivalencia
            - Dividir cada condición de entrada en:

        - Válida
        - Inválida
    - 2. Definir **casos de prueba**
- **Condiciones de borde**
    - La experiencia indica que otorga mejores resultados
    - Acoto los casos en un rango de valores y clasifico. Ejemplo:
        - casos válidos: extremos del rango
        - casos inválidos: valores siguientes a los extremos
- **Ingreso de valores de otro tipo**
    - Forman parte de las clases inválidas
    - Muchas veces son resueltos por el propio IDE
- **Conjetura de errores**
    - **Sospechamos** que algo anda mal
    - Creamos casos de pruebas basados en **situaciones propensas a tener errores**
    - El programador da información más relevante
    - Ideal para componentes
        - complejos
        - “hechos a las apuradas” (algún copy paste por ahí)
        - modificado por varias personas en diferentes momentos
    - No existe una técnica para la conjetura de errores, es basado en la experiencia
- **Integridad de modelo de datos** (para las clases de equivalencia válidas)

  La integridad referencial entre tablas y la cardinalidad de las relaciones definen reglas.

    - De dominio
    - De entidad
    - De relación
- **Diagrama de transición de estados** (solicitud en revisión/rechazada/aceptada)

A veces se solapan estos criterios.

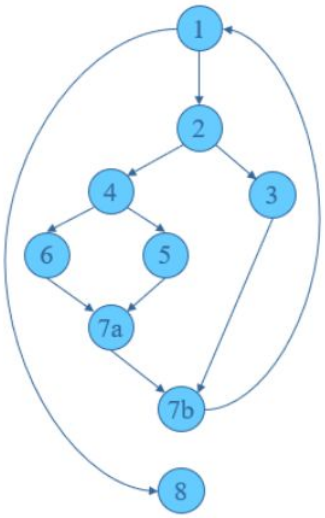
Caja blanca

- Prueba estructural del **software**
- El tester **conoce la estructura interna** y la implementación del software
- El tester determina las entradas y las salidas para ejecutar ciertos **caminos del código**
- Se usa para incrementar el grado de **cobertura de la lógica** interna del componente

Grados de cobertura

Cobertura a nivel

- **Sentencia**
  - prueba cada **instrucción** (sólo una rama del if, no ambas la verdadera *y* falsa)
- **Decisiones**
  - prueba cada **rama** (salida) de un IF o WHILE
- **Condiciones**
  - prueba cada **expresión lógica** de los IF o WHILE
- **Prueba del camino básico**




```
   procedimiento ordenar
1: do while no queden registros
    leer registro;
2:   if campo1 del registro =0
3:   then procesar registro
    guardar en buffer
    incrementar contador
4:   elsif campo2 del registro =0
5:  then reinicializar contador
6:  else procesar registro
    guardar en archivo
7a: endif
    endif
7b:enddo
8: end
```

  - prueba todos los **caminos** independientes
  - las piezas de código se representan a través de un grafo de flujo
  - **complejidad ciclomática** V(g)
    - métrica de la complejidad lógica de un programa
    - depende de la cantidad de caminos independientes
    - V(g) = Aristas - Nodos + 2 (número de regiones del grafo)

Caja gris

Cuando, por ejemplo, ves el código pero llama a una API externa.

Conclusión

- Ninguna técnica de generación de condiciones & casos es completa
  - atacan distintos problemas
  - lo mejor sería combinarlas
- Sin especificaciones de requerimientos se hace muy difícil

TIPOS DE PRUEBA

Prueba unitaria

- Realizada sobre **código bien definido**
- (Generalmente) lo hace el área que **desarrolló** el módulo
- Comienza una vez codificado, compilado y revisado. Se considera parte del desarrollo.
- Los módulos altamente **cohesivos** (buenos límites de su responsabilidad) son más fáciles de probar

Prueba de integración

- No alcanza con verificar el funcionamiento correcto de cada componente por separado
- Se prueban componentes **en su conjunto**
- Permite
    - conectar de a poco partes complejas
    - minimizar la necesidad de programas auxiliares
    - Ver qué pasa si saco un módulo

Tipos

- Incrementales
    - Bottom-up
    - Top-down
    - “Sandwich”
- No incrementales
    - Big Bang

Prueba de aceptación de usuario

- Realizada por los **usuarios**
- Basada en **requerimientos**
    - verifica que se ajusten a las **necesidades** del usuario
- De **caja negra**

Pruebas no funcionales

Prueba de volumen

- Verifica que el sistema soporta los **volúmenes máximos** definidos en
  - almacenamiento
  - procesamiento

Prueba de performance

- Orientada a verificar que el sistema soporta los tiempos de respuesta requeridos.
- Se realizan de la mano de las de volumen.

Prueba de stress

- Similar a la prueba de volúmen pero **excediendo** los máximos definidos
- Se intenta buscar el punto de ruptura
- Simula ser una situación no prevista: desenchufo el server a ver qué pasa

Prueba de seguridad (Pen testing / Ethical hacking)

- Orientado a probar los requerimientos de seguridad.
- Ejemplo: Penetration test. Se simula que ingresa un intruso (interno o externo).
- Puede ser automática o manual

Prueba de usabilidad

- Orientado a probar los atributos de usabilidad en los requerimientos.
- Prueba de campo/observación.

Prueba de regresión

- Verifica que los nuevos cambios introducidos no afecten lo anterior
  - Se prueba eso anterior, NO lo nuevo

Prueba de humo/Smoke test

- Test de alto nivel, no se entra en detalles y no se planifica

Prueba Alfa y Beta

- Realizada por el usuario
  - es económico

        - no siempre se puede hacer
        - Alfa
            - lo prueba en mis instalaciones
        - Beta
            - lo prueba en sus instalaciones (usuario final)
- Software
    - se prueba una primera versión
    - plagado de defectos

Exploratory testing

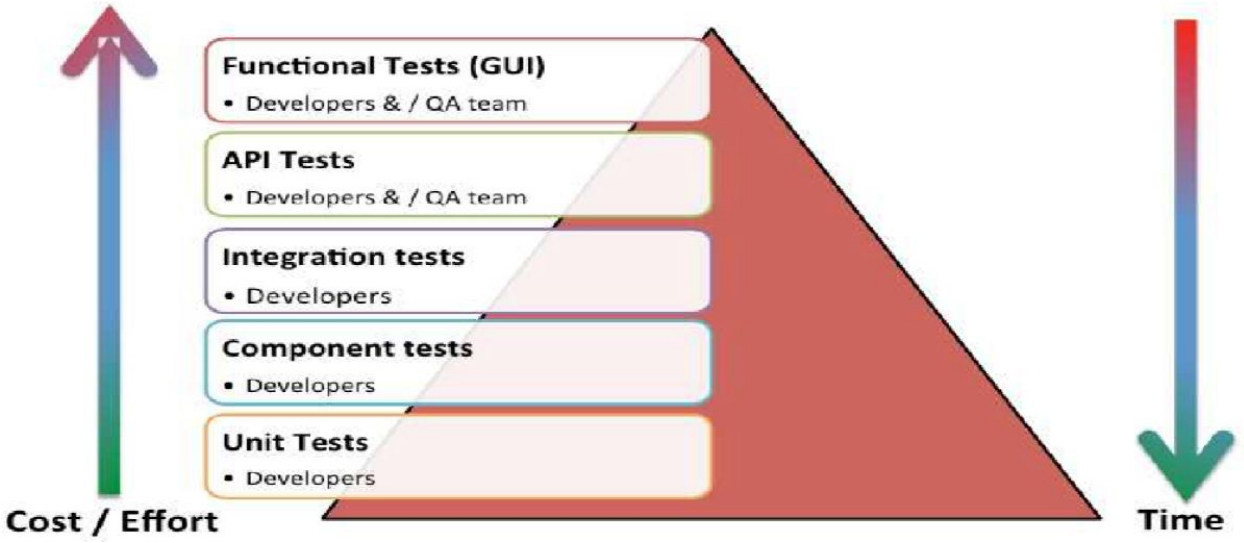
- Guionado/script driven o no guionado/unscripted

MODELOS DE CICLOS DE TESTING

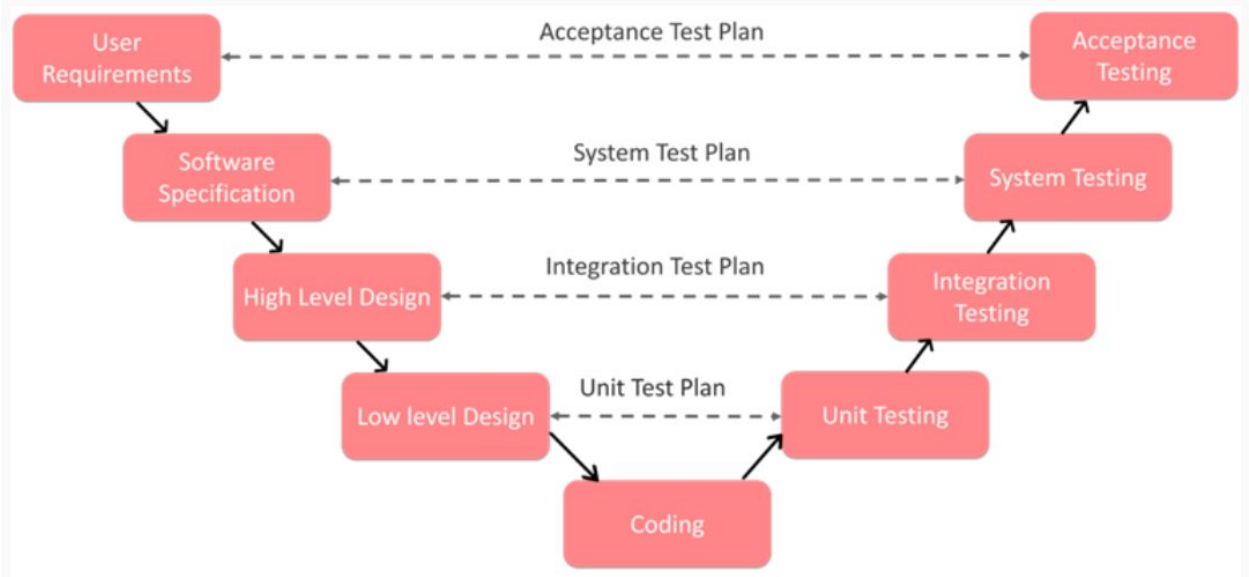
Clasificación de las pruebas

Conocimiento	Tipo de ejecución	Tipo de prueba	Alcance o nivel
• Caja blanca • Caja negra • Caja gris	• Automáticas • Manuales • Semi-automatizadas	• Funcionales • No funcionales	• Unitarias • De integración • De sistemas • De aceptación de usuario

Pirámide del testing



Modelo de ciclo de vida V/V model



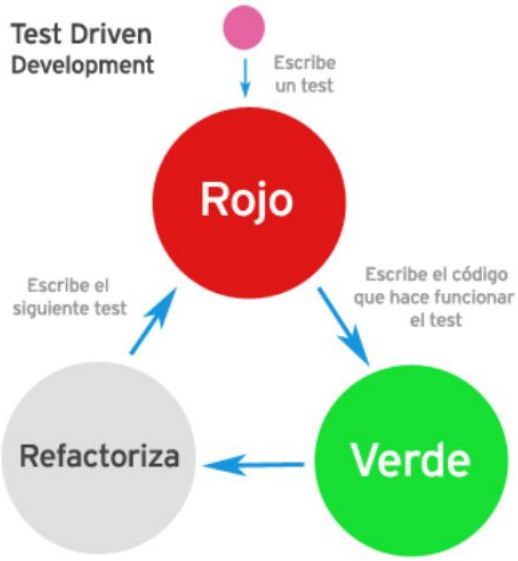
Cuadrantes, niveles y tipos de testing

	Orientado al negocio	**Orientado a la tecnología**
Soportan al equipo	*Automatizados y manuales* • Functional tests • Story tests • Prototipos • Simulaciones	*Automatizados* • Unitarios • Por componente • De integración
Miran al producto	*Manuales* • De aceptación de usuario • Alpha-Beta • UX / UI • Exploratory • Scenarios	*Semi-automatizados con herramientas* • De volumen • De seguridad • De perfomance • "...ility"

Agile testing (TDD?)

Pros

- Aplicaciones
  - + modulares
  - + flexibles
  - + escalables
- Código
  - - acoplado
  - + mantenible
- Mayor cobertura de testing
- Refactoring con bajo riesgo
- Soporta regresión de bugs



Contras

- Se repiten los mismos errores tanto en el código como en el test case
  - Si el que escribe los tests es el que escribe el código
- No apto para algunas funcionalidades
- Difícil mantenimiento
- Riesgo de falso nivel de confianza

CONCLUSIONES PPT Testing

- La definición del resultado esperado es esencial
- Promover el TDD
    - Evitar probar mi componente
    - Desventaja: gran **sesgo** al hacer los test. Se evita con **peer reviews**
- Suponer que se encontrarán errores
- Probar es un desafío intelectual
- Las pruebas no mejoran el software
- El buen diseño contribuye no solo a mejores pruebas, sino también a la corrección y mantenimiento de los componentes
- Lo más barato para encontrar y eliminar defectos es no introducirlos
- La automatización de pruebas ayudan pero no son suficientes

WHEN TWO EYES ARE NOT ENOUGH

Peer reviews

Peer reviews según formalidad

1. **Inspection** (más formal)
    a. Proceso muy formal con moderador, equipo de inspección y de registro
    b. Un participante explica cada pedazo de código
    c. Las inspecciones son **mucho** más efectivas para encontrar defectos que las otras técnicas
2. **Team Review**
    a. Le dan el producto a los que revisan para que se lo estudien antes de la reunión
    b. Roles más relajados que en la inspección (el dev podría ser el moderador por ejemplo)
    c. El moderador pregunta si tienen algún comentario sobre lo que leyeron
3. **Walkthrough**
    a. El autor le describe el producto a un grupo para que opinen
4. **Pair Programming**
    a. Dos devs trabajan en el mismo producto en una sola compu
    b. Puede usarse para cualquier entregable, no sólo código.
5. **Peer Desckcheck**
    a. Solo una persona revisa el código (además del autor) y le cuenta qué onda
    b. Depende de la voluntad y conocimiento de esa persona
    c. **Passaround**
        i. Lo mismo pero le enviás el código a varios
        ii. No brinda la sinergia de las reuniones
6. **Ad Hock Review** (menos formal)
    a. Le preguntas qué onda a alguien

Características de los métodos formales

- **Objetivos** definidos
- Participación de un **equipo** entrenado
- **Moderador**
- **Roles** y responsabilidades específicos
- **Procedimientos documentados** para la revisión
- Los resultados se **reportan** a la **gerencia**
- **Seguimiento** de los defectos encontrados hasta su cierre

- **Registro** del proceso para mejorar la revisión
- Data de calidad para mejorar el desarrollo del producto

Actividades de cada revisión

Tipo	Planeamiento	Preparación	Reunión	Corrección	Verificación
Inspection	Sí	Sí	Sí	Sí	Sí
Team Review		Sí			No
Walkthrough		No			No
Pair Programming		No	Continua		Sí
Peer Deskcheck, Passaround	No	Sí	Probablemente		No
Ad Hoc Review		No	Sí		No

Cómo elegir cuál usar

- Según el riesgo por:
  - La probabilidad de que tenga un error
  - El impacto que tendría un error
- Factores del riesgo:
  - Uso de nuevas tecnologías
  - Algoritmos complejos
  - Mucha presión en el dev
  - Gente sin experiencia
  - Componentes importantes con muchas maneras de fallar
  - Componente en que se va a basar el resto del desarrollo
  - Componentes reusables
  - Componentes que afectan varias partes del producto

También se puede elegir cuál usar según qué quieras lograr con la revisión (tablita en el paper): encontrar defectos de implementación, validar conformidad con las especificaciones/estándares, verificar la completitud del producto, asegurar la mantenibilidad, probar la calidad de productos críticos, recolectar información para la mejora de los procesos o medir la calidad del documento.

TEST ORACLES

Cómo explicar que algo es un bug, por más que no esté explicitado en los requerimientos

Test oracles: el mecanismo por el que reconocemos un problema.

Heurística “HICCUP”

El producto (desde el punto de vista del tester) debe tener consistencia en su:

- **H**istory: ¿Algo cambió sin aviso?
- **I**mage: ¿Se ve profesional?
- **C**omparable Product: ¿El producto similar tiene este comportamiento?
- **C**laims: ¿Coincide con lo que se habla sobre producto?
- **U**ser Expectation: ¿El usuario espera este comportamiento?
- **P**roduct: ¿Hay cambios en la terminología / cómo funciona / look and feel dentro del mismo producto?
- **P**urpose: ¿Es consistente con el propósito del software en sí? Por ejemplo: valores negativos en el tamaño de letra en Word

Usando estos puntos podemos expresar por qué pensamos que algo es un bug.

Preguntas de parciales:

1. ¿Existe alguna relación entre un software baseline y la función de Software Configuration Change Control?

Sí, un cambio o conjunto de cambios considerados y analizados por SCCC pueden partir de un baseline y eventualmente formar parte, con su nuevo estado, de un nuevo baseline.

2.

Un error en el código ya resuelto vuelve a aparece	Config identification (estaban mal identificadas las relaciones entre los ICs)
Necesito saber cuántos cambios tuvo un determinado componente durante el último año	Status and Accounting
Durante el período de fin de año necesito asegurarme que no se implementen cambios en el software hasta cerrar los balances contables	Software Configuration Change Control
Necesito saber quién fue el que introdujo el último cambio en un componente	Status and Accounting
Verificar si un cambio cumple con la funcionalidad que se pretende implementar	Config Audit
Encontrar un componente en una biblioteca donde hay muchos componentes similares	Config Identification