Resumen Ingeniería de Software – Segundo Parcial – 1C2022 - Dalceggio

Unidad N°5: Software Configuration Management"

Problemas en el Desarrollo de Software

- Problemas "mas" de ingeniería que son comunes a todos los equipos de trabajo.
- Problemas cuando empezamos a gestionar, hacer funcionar código en muchos lugares.
- SCM -> Gestionar cambios al Software.

Configuración

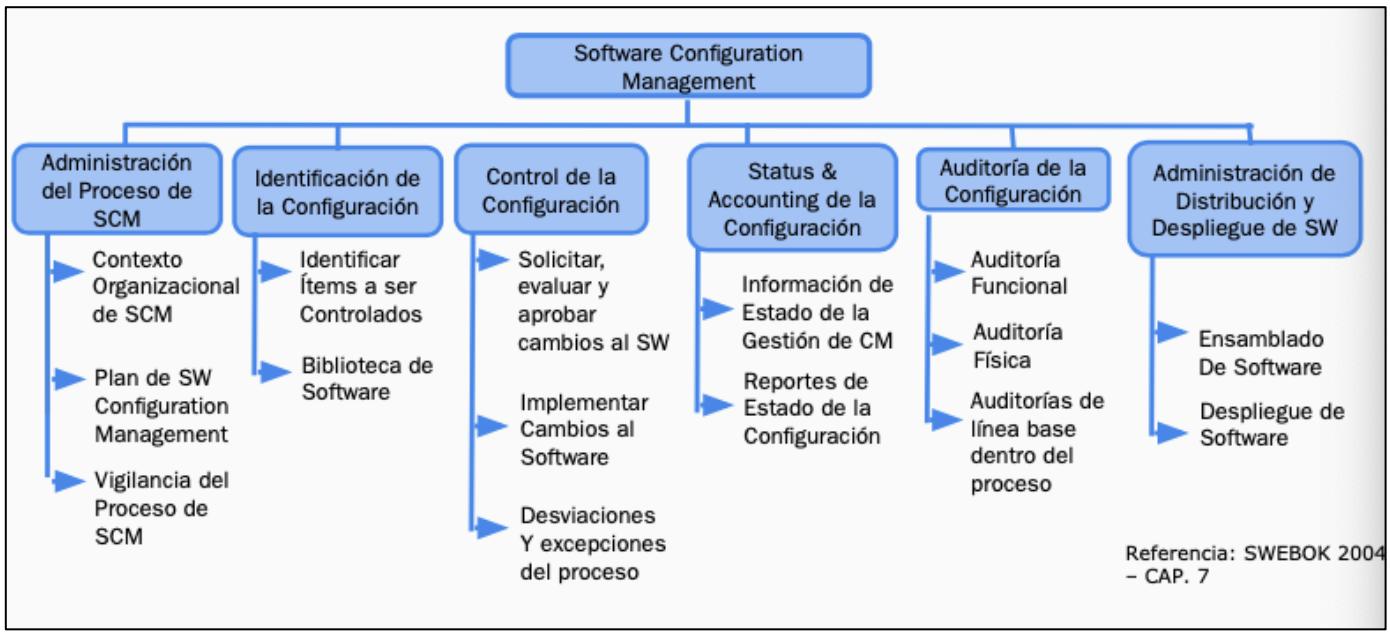
- **Ítem de Configuración**
  - **Cualquier elemento involucrado en el desarrollo del producto y que está bajo el control de SCM**. (Código, servers, scripts, modelos, bases de datos). **No todo necesita estar bajo SCM**.
  - Cada **ítem se conoce información sobre su configuración** (nombre, versión, fecha de creación y autor).
  - Cualquier elemento en el desarrollo de software que se debe gestionar.
- **Configuración**
  - Conjunto de **todos los componentes fuentes que son compilados en un ejecutable consistente**.
  - Conjunto de todos los "ítems de configuración**". Definen una "versión".**
  - **Todos los componentes, documentos e información de su estructura que definen una versión determinada** del producto a entregar.

Gestión de la Configuración de Software (SCM)

- **Disciplina orientada a administrar la evolución de productos, procesos y ambientes**.
- **Propósito es establecer y mantener la integridad de los productos del proyecto de software a lo largo del ciclo de vida del mismo**.
- Acompaña la actividad de cambio con actividades de control.
- Involucra para una configuración:
  - Identificarla en un momento dado.
  - Controlar cambios sistemáticamente.
  - Mantener su integridad y origen.
- **Acompaña la actividad de cambio con actividades de control.**

SCM de acuerdo con SWEBOK

- **SWEBOK: define todos los puntos para establecer una buena SCM. Facilita desarrollo y actividades de implementación de cambios.**
- **Pasos de SCM se basan en el SWEBOK.**
-



Pasos de SCM

0. Administración del Proceso de SCM

- **Etapa previa al arranque, donde se debe comprender el contexto organizacional.**
- **Cuál es mi punto de partida a la hora de arrancar con SCM. Cuál es el contexto organizacional.**
- Comprobar si existen procesos de SCM en la organización, ver que herramientas hay disponibles.

1. Identificación de la Configuración

- **Definir qué elementos (Ítems de Configuración) estarán controlados por la gestión de configuración.**
  - **Identificar Ítems de Configuración.**
  - **Identificar Relaciones entre Ítems de Configuración.**
- **No todo necesita estar bajo SCM.** Es necesario contar o definir con guías para saber que elementos son parte y cuales no.
- **Define momentos o condiciones para establecer una línea base o bien para liberarla.** Es decir, establecer un baseline (versión aprobada de la configuración) o hacer un reléase.

2. Control de la Configuración

- Teniendo la configuración identificada, debemos **establecer un orden para hacer los cambios en el software con la idea de mantener la integridad**.
- **Asegurar que los ítems de configuración mantienen su integridad ante los cambios** a través de:
    - **Identificación del propósito del cambio**.
    - **Evaluación del impacto y aprobación del cambio**.
    - **Planificación de la incorporación del cambio**.
    - **Control de la implementación del cambio y su verificación**.
- Establecer procedimiento de control de cambios y controlar el cambio y la liberación de ICs a lo largo del ciclo de vida.
- Analizar el impacto del cambio, autorizarlo y realizar el seguimiento de los pedidos de cambio hasta su cierre. Solicitud de cambio debe ser formalmente registrada.

3. Status & Accounting de la Configuración

- **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.
- **Debemos pode contestar por todos los cambios que se realizaron al sistema**.
- **Se debe informar periódicamente con reportes a todos los grupos afectados**.

4. Auditoria de la Configuración

- **Realizar una verificación del estado de la configuración a fin de determinar si se están cumpliendo los requerimientos especificados**.
- Puede ser ejecutada con **diferentes niveles de formalidad**:
    - **Revisiones informales** basadas en chechklists.
    - **Pruebas exhaustivas** de la configuración que son **planificadas**.
- **Tipos de Auditoria**:
    - **Auditoria Funcional**: verificar el cumplimiento de los requerimientos.
    - **Auditoria Física**: verificar que la configuración del producto se corresponda con la estructura especificada. Ver el detalle de la implementación. Si cambio lo que dijeron.
    - **Auditoria de Proceso**: verificar que se haya cumplido el proceso de SCM en todo el proyecto.

5. Administración de Distribución y Despliegue de Software

- Luego de hacer los cambios -> **Administración, identificación y distribución de un producto**.
- **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.
- Se divide en **2 partes**: **Software Building y Software Release Management.**

Proceso de Software Configuration Management

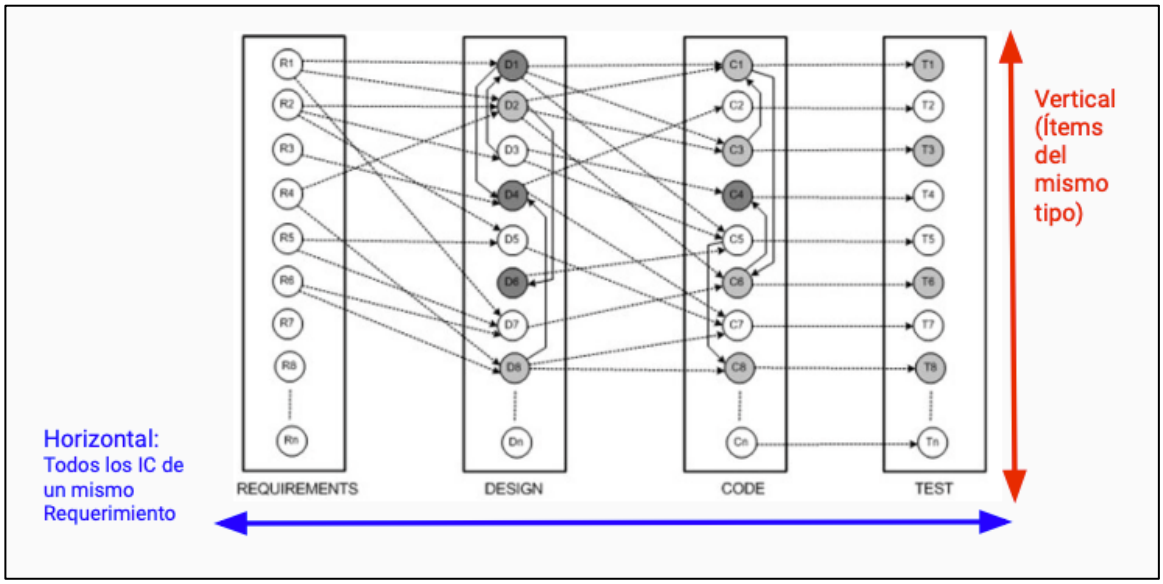
- En las compañías actualmente se presenta un problema de SISOP: "*Hicimos un merge del código y funciona, pero rompe en producción*".

Planificar el Proceso de SCM

- **Plan puede ser definido por proyecto o a nivel organizacional y luego adaptarlo al proyecto**.
  1. Listar los ítems de configuración, y en qué momento ingresan al sistema de CM.
  2. Definir estándares de nombres, jerarquías de directorios y estándares de versionamiento.
  3. Definir políticas de branching y de merging.
  4. Definir los procedimientos para crear "builds" y "releases".
  5. Definir reglas de uso de la herramienta de CM y el rol del administrador de la configuración.
  6. Definir el contenido de los reportes de auditoría y los momentos en que se ejecutan.

Definiciones Básicas

- **Baseline (Línea de Base)**
  - **Estado de configuración de un conjunto de ítems en el ciclo de desarrollo, que puede tomarse como punto de referencia para una siguiente etapa del ciclo**.
  - Se establece porque se verifica que esa configuración satisface algunos requerimientos.
  - Para una baseline, hay un conjunto de cambios a los ICs con respecto a la baseline anterior.
  - Baseline y Release no son lo mismo.
- **Branch (Rama)**
  - **Línea de desarrollo separada**.
  - **Estas líneas utilizan la baseline de un repositorio existente como punto de partida**.
  - Permite a los miembros del equipo trabajar en múltiples versiones de un producto, utilizando el mismo set de ICs.

- **Trazabilidad entre Ítems de Configuración**
  - El Configuration Control Board (CCB) se fija acá que hay que tocar si quiero cambiar cosas.
  - 

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, etc.
- **Además del ejecutable, comprende la administración, identificación y distribución de un producto.**

Software Building

- **Los Builds**:
  - **Deben ser automáticos**.
  - **Deben permitir la generación de reportes**. Cada build genera un reporte del estado del mismo.
- **Beneficios para el negocio**:
  - **Reducen la cantidad de defectos**.
  - **Mejora la reproducción de problemas y trazabilidad**.
  - **Mejora la performance del equipo de desarrollo**.
- **Tipos de Builds**:
  - **Local**: developer lo hace de manera local y corre las pruebas unitarias.
  - **Integration**: generar el entorno completo para las pruebas de integración.

- **Nightly**: ejecutar la construcción en forma diaria y generar reportes con información sobre el build.
- **Release**: cuando se decide crear una nueva versión a ser liberada.

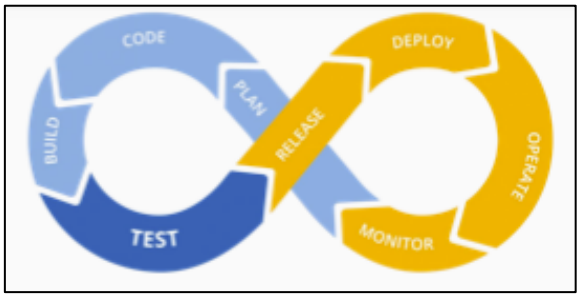
Software Deployment

- **Una vez generada la versión o release, debemos buscar el mecanismo más efectivo para hacer despliegue controlado por los distintos usuarios**.
- Para **determinar un modelo de deployment**:
  - Evaluar que usuarios deberán recibir los cambios.
  - Evaluar en que forma se deberá hacer el despliegue.
  - Evaluar riesgos del despliegue y como se minimizan.
  - Aprobación por el negocio y/o área de QA.
  - Evaluar quienes realizaran el deployment de la versión definida.

Build & Deployment Pipelines

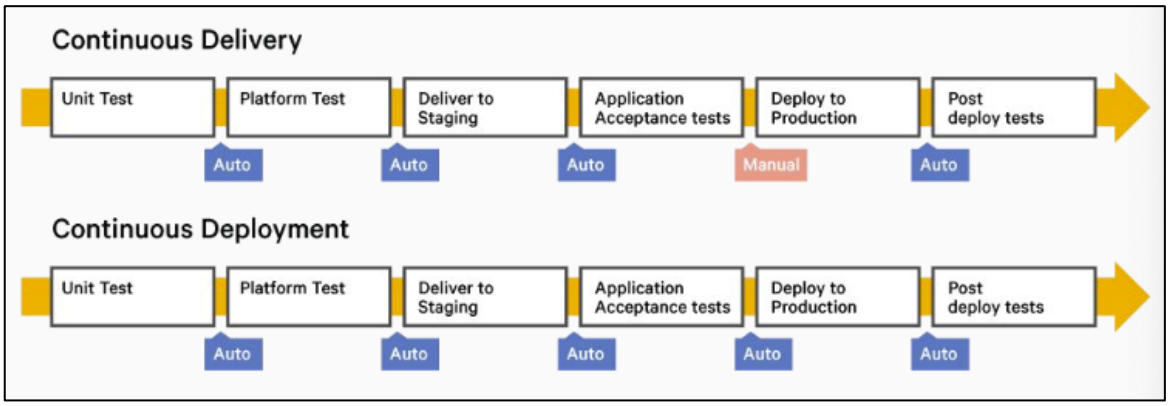
- **Manifestación automatizada del proceso para llevar el software desde la aprobación del cambio de SCM hasta que llega a manos de los usuarios**.
- **Incluye el camino complejo desde que se compila y construye el software, seguido del progreso a través de varias etapas de testing y deployment**. Requiere colaboración entre individuos y equipos.
- **Gated Commits**: buena práctica de construcción que resuelve el problema puntual de mantener master en estado integro tras hacer un cambio. Valida el cambio antes de hacer merge.

Integración Continua y Despliegue Continuo (CI/CD)



- **Integración Continua (CI)**
  - **Realizar integración de código al menos una vez por día o más, para minimizar problemas del código**.
  - Buenas Prácticas:
    - Repositorio de código único.
    - Automatizar el proceso de build.
    - Hacer el build testeable automáticamente.
    - Todo commit debe construirse por una herramienta de integración y no por el dev.
    - El build debe ser rápido.

- **Despliegue Continuo (CD)**
    - **Conjunto de prácticas que permiten asegurar que el software puede ser desplegado en producción rápidamente y en cualquier momento**.
    - Lo logra haciendo que cada cambio llegue a un entorno de staging, donde se corren pruebas exhaustivas.
    - Puede **pasar a producción manualmente cuando alguien apruebe el cambio**.
- **Son procesos complementarios que funcionan de la mano**.
- **No confundir Despliegue Continuo** (Continuos Delivery) **con Deployment Continuo** (Continuos Deployment). Este **es el paso siguiente al Delivery en donde el despliegue a producción se realiza en forma automática por un proceso y no por una persona**.
- **Comparativa**:



DEVOPS

- Vienen a terminar con la cultura de separación y falta de comunicación y colaboración entre los developers y la gente de operations.
- **DevOps: conjunto de prácticas destinadas a reducir el tiempo entre el cambio en un sistema y su pasaje a producción, garantizando la calidad y minimizando el esfuerzo**.
- **Es la combinación de desarrollo y operaciones, normalmente en un equipo. Permite definir la estructura describiéndola, codeándola en las especificaciones**.
- **Practicas**
    - Planificación del Cambio
    - Coding & Building
    - Testing
    - Release & Deployment
    - Operation & Monitoring

- **CALMS (Filosofía de Trabajo)**
    - **Cultura**: ser dueños del cambio para mejorar la colaboración y comunicación.
    - **Automatización**: eliminar el trabajo manual y repetitivo llevando a procesos repetibles y sistemas confiables que reduzcan el error humano.
    - **Lean**: remover la burocracia para tener ciclos más cortos y menos desperdicio.
    - **Métricas**: medir todo, usar datos para refinar los ciclos.
    - **Sharing**: compartir experiencias de éxito y falla para que otros puedan aprender.

Paper: "*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*". **Objetivo**: lograr la integridad del proyecto.
- **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.
    - 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?

Unidad N°6: "Testing"

Software de Calidad – Aseguramiento Calidad

- **Software de Calidad**: es aquel que **cumple con los requisitos y al ser usado no tiene fallas**.
- **Aseguramiento de Calidad**:
  - **Asegurar la calidad del proceso definido para cumplir con los requerimientos técnicos. Asegurar el cumplimiento del proceso para construir algo.** Todo el proceso.
  - **Se diferencia del control de calidad. Asegurar y brindar confianza sobre lo que estoy construyendo.** Control de calidad es sobre algo que ya está hecho.
  - **Patrón planificado y sistemático de todas las acciones necesarias para brindar una adecuada confianza que un producto cumple con los requerimientos técnicos establecidos**.
  - Prueba de Software es una de las actividades involucradas en aseguramiento de calidad.
- **Asegurar vs Controlar**:
  - La calidad **no puede inyectarse al final**. Se piensa desde el comienzo y no se agrega al final.
  - Calidad del producto **depende de tareas realizadas durante el proceso**.
  - **Detectar errores en forma temprana ahorra esfuerzos, tiempo y recursos.**

Objetivo del Testing

- **Objetivo principal consiste en encontrar fallas en el producto, de manera eficiente y eficaz**.
- **Eficiente**:
  - Hacerlo lo más **rápido** posible.
  - Hacerlo lo más **barato** posible.
- **Eficazmente**:
  - Encontrar la **mayor cantidad de fallas.**
  - **No detectar fallas que no son** (Falsos Positivos).
  - **Encontrar las fallas más importantes**.

Prueba de Software (Testing)

- **Definición según IEE**:
  - Actividad en la cual un sistema es ejecutado bajo condiciones específicas.
  - Resultados son observados y registrados para realizar una evaluación.
  - Proceso/Actividad dinámica al detectar errores en forma temprana.

- **Definición Usual**:
    - Actividad donde se prueba un componente con el objetivo de producir fallas.
    - Una prueba es exitosa si encuentra fallas.
    - Compara los resultados obtenidos con los esperados.

Proceso de Prueba de Software

- **Pasos (Simplificado**):
    1. Diseñar y Construir Prueba: debo definir una entrada/estado inicial valido y el resultado esperado que busco obtener. Resultado genérico, no números o valores.
    2. Ejecuto la Prueba.
    3. Comparo Resultado Obtenido: comparo el resultado obtenido contra el determinado.
- Debo tener clara la funcionalidad que deseo testear. Es lo principal para saber que se debe testear. Definir entradas y el resultado esperado.
- Proceso de pruebas debe estar detallado.

Incidente, Falla, Equivocación y Defecto

- **TODO LO QUE ENCUENTRO ES INCIDENTE**. Después es falla, error, etc. **No todo incidente es falla.**
- **Defecto es lo que me origina la falla**. La falla es la manifestación del defecto.
- **Incidente**: toda ocurrencia de un evento que sucede durante la ejecución de una prueba de software que requiere investigación. No toda incidencia es una falla.
- **Equivocación**: acción humana que produce un resultado incorrecto.
- **Defecto**: paso, proceso o definición de dato incorrecto. Ausencia de cierta característica.
- **Falla**: resultado de ejecución incorrecto. Es el producido por el SW distinto al resultado esperado.

Conceptos Relacionados

- **Condiciones de Prueba**: son descripciones de situaciones que quieren probarse ante las que el sistema debe responder.
- **Casos de Prueba**: son lotes de datos necesarios para que se dé una determinada condición de prueba.
- Crear **condiciones es un proceso "creativo**", mientras que **crear casos de prueba es un proceso "laborioso"**.
- **Criterios de Selección**: es una condición para seleccionar un conjunto de casos de prueba. De todas las combinaciones posibles, solo seleccionamos algunas en base a la probabilidad de encontrar defectos.

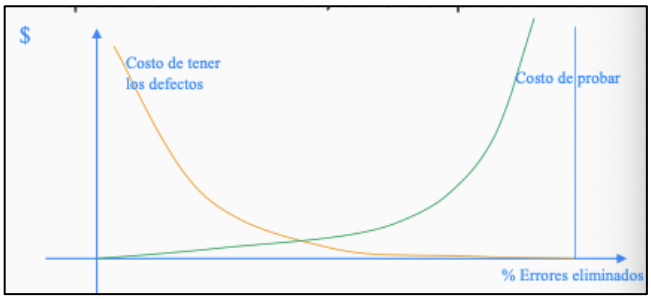
- **Partición**: todos los posibles casos de prueba los dividimos en clases. Todos los casos de una clase son equivalentes entre sí, es decir que detectan los mismos defectos. Con solo ejemplos de cada clase ya cubrimos todas las pruebas, por lo que el éxito está en hacer una buena selección de la partición.

Depuración

- **Depuración es eliminar un defecto que posee el Software. No es una tarea de prueba, aunque es consecuencia de ella.**
- **Depuración puede ser fuente de introducción de nuevos defectos**.
- **Proceso de Depuración:**
  1. **Detectar**: prueba detecta la falla (efecto) de un defecto. La depuración detecta el defecto (causa).
  2. **Depurar**: eliminar el defecto. Encontrando la razón del mismo, planteando una solución y aplicándola.
  3. **Volver a Probar**: asegurar que sacamos el defecto y que no hemos introducido otros.
  4. **Aprender para el Futuro**.

Economía del Testing

- ***¿Hasta Cuándo Probar?***
  - Concepto que se debe definir ya que se puede invertir mucho esfuerzo y tiempo en probar.
  - Ya que nunca se va a poder demostrar que un SW es correcto, continuar probando es una decisión económica.



- ***¿Cuándo Detengo la Prueba?***
  - Depende de lo que pruebo, hay punto de quiebre entre costo de probar, costo de tener defectos y los errores eliminados. Depende de la industria y del negocio.
  - Estrategias:
    - Software supera mi conjunto de pruebas diseñado.
    - Se acepta cierta cantidad de fallas no criticas (Good Enough).
    - Cantidad de fallas detectadas es similar a las estimadas.
- ***¿Cómo Abarato la Prueba?***
  - Para abaratar y acelerar el testing, sin degradar su utilidad, se debe diseñar un software apto para ser testeado.
  - Diseño modular, ocultamiento de información, puntos de control y programación no egoísta.

Padecimiento de Requerimientos (Debilidades)

- **Ambigüedad**: cuando para dos personas es distinto significado.
- **Incorrectitud**: requerimiento mal definido por lo que no va.
- **Inconsistencia**: ante un mismo evento, obtengo dos resultados diferentes.
- **Incompletitud**: requerimiento esta incompleto.
- **Mejor manera de atacar las 4 debilidades es ver como los voy a probar**. Una vez que tengo los requerimientos definidos, genero como voy a testear cada uno.
- **Cuando pienso como voy a probar lo que tengo escrito, me doy cuenta de las inconsistencias.**

Unidad N°6: "Prueba Funcional" (*Caja Negra*)

Prueba de Caja Negra

- **Prueba funcional, producida por los datos, o producida por la entrada/salida**.
- **Prueba lo que el software debería hacer**. **Se basa en la definición del módulo a probar** (definición necesaria para construir el módulo).
- **Nos desentendemos completamente del comportamiento y estructura interna del componente**. No tiene en cuenta implementación ni arquitectura.
- Usuarios generalmente hacen pruebas de caja negra.

¿Qué Hacemos?

- La prueba de caja negra exhaustiva es imposible de realizar. Tendría que probar todos los valores posibles de todos los datos de entrada (infinito en general).
- Se debe seleccionar un subconjunto de datos de prueba que representen conjuntos mayores de datos posibles. Suponemos que la prueba de un valor representativo es equivalente.
- **Condiciones de Prueba**: son descripciones de situaciones que quieren probarse ante las que el sistema debe responder.
- **Casos de Prueba**: son lotes de datos necesarios para que se dé una determinada condición de prueba.

Criterios de Selección

- **Partición en Clases de Equivalencia**:
    - **Proceso incluye dos pasos**: identificar las clases de equivalencia y definir los casos de prueba.
    - La identificación de clases de equivalencia se hace dividiendo cada condición de entrada en dos grupos. **Clases Validas y Clases Invalidas**.
        - **Por Condición de Entrada**:
            - Rango de valores (una valida y dos invalidas).
            - Conjunto de valores (una valida y una invalida).
            - "Debe Ser" (una valida y una invalida).
        - **Condiciones de Borde**:
            - Rango de valores (casos válidos para los extremos del rango e inválidos para los valores siguientes a los extremos).
            - No agregan nuevas condiciones/clases. Casos extremos, siempre incluirlos.
        - **Clases Invalidas**: no son del mismo tipo que las clases invalidas y validas. Prueban el ingreso de valores de otro tipo. Agregar también condiciones cruzadas.
- **Prueba de Conjetura de Errores**:
    - Prueba de Sospechas. Se sospecha que algo funciona mal.
    - Proceso muy efectivo y formalizado a partir del análisis de las fallas.
    - El programador es quien otorga la información más relevante.
    - Creatividad juega un papel clave.

Unidad N°6: "Prueba Estructural" (*Caja Blanca*)

Prueba Estructural

- **Prueba lo que el software hace. Se basa en cómo está estructurado el componente internamente y su definición**.
- Usada para incrementar el grado de cobertura de la lógica interna del componente.
- **Tengo una entrada y espero una salida, pero sabiendo los componentes que resuelven.**
- **NO es probar líneas de código**. Las líneas de código siempre funcionan. **Tengo que entender como está construido y si cumple con lo pedido**.
- **Anexo las condiciones de prueba en base a la arquitectura. Sabiendo la implementación, agrego condiciones de prueba**.

Grados de Cobertura

- **Cobertura de Sentencias**
    - Prueba cada instrucción. A partir de mis casos de prueba, una vez que corren, miro si todas las líneas de código fueron barridas/ejecutadas por el test.
    - Busca dotar de condiciones que generen casos, en busca de que todas las líneas de código hayan sido ejecutadas.
- **Cobertura de Decisiones**
    - Busca probar las decisiones dentro del código. Me obliga a probar todas las condiciones de negocio.
    - Cobertura de decisiones es más completo que cobertura de sentencias, prueba cada salida del if o while.
- **Cobertura de Condiciones**
    - Busca probar todas las condiciones que hay dentro de una condición. Prueba cada expresión lógica de los IFs o Whiles.
- **Cobertura Camino Básico**
    - Se representa el flujo de control de una pieza de código a través de un grafo de flujo.
    - Prueba todos los caminos independientes.
    - **Complejidad Ciclomatica**
        - Métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa. Cantidad de caminos independientes.
        - Me ayuda a probar la cobertura de decisiones. Plantea todas las combinaciones posibles.

Unidad N°6: “Prueba de Caja Gris”

- Prueba que combina elementos de la caja negra y caja blanca. Conocimiento parcial no total (si no es caja blanca).
- No es caja negra porque se conoce parte de la implementación o estructura interna y se aprovecha ese conocimiento para generar condiciones y casos que no se generarían naturalmente en una prueba de caja negra.
- Pruebas de caja gris prueban como si fuera caja negra, pero suman condiciones y casos adicionales derivados del conocimiento parcial de la operación e interacción de los componentes.
- Sucede con mayor frecuencia ya que cada vez es más raro ver entero como todo esta implementado.

Unidad N°6: “Tipos de Prueba”

Prueba Unitaria

- Se realiza sobre una pieza de código claramente definida.
- Generalmente la realiza el área que construyo el módulo.
- Se basa sobre el requerimiento / especificación y no lo que el programador codeo.
- Comienza una vez codificado, compilado y revisado el módulo.
- Los componentes altamente cohesivos son más fáciles de probar que los que están todos acoplados.

Prueba de Integración

- Orientada a verificar que las partes de un sistema funcionan bien en conjunto.
- Junto piezas de código en base a una estrategia definida. Lo más importante es como vamos a integrar, por lo que la estrategia es lo primero y más importante.
- **Tipos**:
    - Incrementales: bottom-up, top-down y sándwich.
    - No Incrementales: big bang.

Pruebas de Aceptación de Usuario (UAT)

- Prueba realizada por los usuarios para verificar que el sistema se ajusta a sus requerimientos.
- Las condiciones de pruebas están basadas en el documento de requerimientos. Lo que dije que quiero que haga el sistema.
- Es una prueba de caja negra.

Prueba No Funcional

- Las pruebas no funcionales buscan comprobar la satisfacción de los requerimientos no funcionales de software.
- Pruebas que ejecuto no para saber si hace lo que debe, si no si está cumpliendo las condiciones sobre las que debe hacerlo.
- **Clasificación**: performance, volumen, stress, seguridad, usabilidad, escalabilidad.

Prueba de Regresión

- Orientada a verificar que, luego de introducido un cambio en el código, la funcionalidad original no ha sido alterada y se obtengan comportamientos no deseados o fallas en módulos no modificados.
- Prueba donde se prueban el resto de las funcionalidades después de agregar una nueva.

Prueba de Humo (Smoke Test)

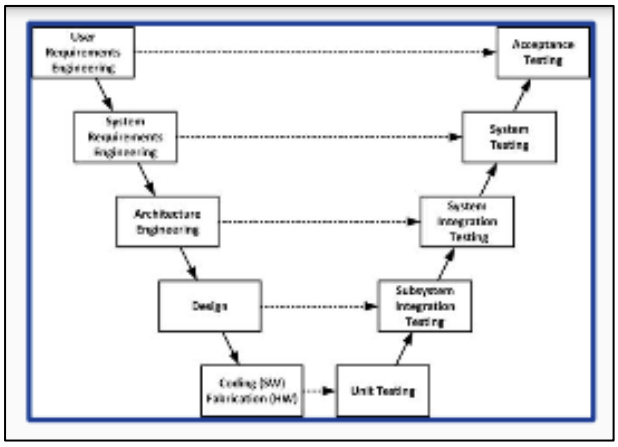
- Orientada a verificar de una manera muy rápida que en la funcionalidad del sistema no hay ninguna falla que interrumpa el funcionamiento básico del mismo.
- Prueba rápida para ver si funciona lo super básico. Es un test de muy alto nivel.

Prueba Alfa & Beta

- Se entrega una primera versión al usuario que se considera esta lista para ser probada por ellos.
- **Prueba Alfa**: se hace in-house. Prueba dentro del dominio de la organización.
- **Prueba Beta**: lo hace el público en general. Crowd Testing. Busca variación en el tipo de prueba por fuera del dominio de la organización.

Prueba V-Model

- Prueba basada en el método en cascada. Adelantarse a identificar las cosas en un momento temprano.
- **Identifico**
    - Requerimientos.
    - Arquitectura.
    - Especializado el Detalle.
    - Codeo.
- **Hago**
    - Codeo.
    - Prueba Unitaria.
    - Prueba de Integración.
    - Prueba UAT.



Prueba de Exploración – Exploratory Testing

- Es el aprendizaje, diseño y ejecución de la prueba en forma simultánea.
- Es sin guion. Condiciones y casos no son diseñados y documentados previamente para ser ejecutados.
- El tester va decidiendo tácticamente que es lo mejor de acuerdo al conocimiento que va adquiriendo de la ejecución de un test. Depende mucho de la skill del tester y su experiencia. El tester es responsable en todo momento de la decisión del camino a tomar.
- **Etapas**:
    1. Reconocimiento y Aprendizaje: identificar toda la información que nos permita conocer que es lo más importante a probar y como hacerlo.
    2. Diseño: crear una guía draft para probar.
    3. Ejecución: ejecutar los casos y registrar los resultados.
    4. Interpretación: obtengo conclusiones.
- **Charter**: define la misión de la sesión de testing. No es un plan detallado, define que se debería testear, como y que tipo de defectos se buscan.

Testing Guionado y No Guionado

- **Guionado**: se confeccionan las condiciones y casos tempranamente para luego ser ejecutados. Muchas veces, cada ciclo de prueba se vuelven a repetir las condiciones y casos, y no se retroalimentan.
- **No Guionado**: se confeccionan las condiciones y casos a medida que se aprende de las distintas ejecuciones, refocalizando las pruebas si fuera necesario. Las ideas ocurren “on the fly”.

Unidad N°6: “Métricas de Testing”

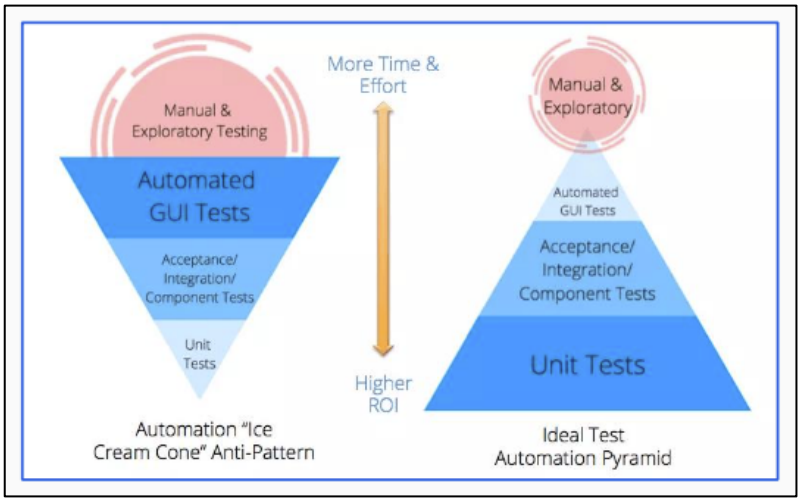
- **Difícil evaluar a un equipo de testing. Por lo que necesito métricas para comparar y dar valores.**
- **Métricas sugeridas a la hora de analizar los tests**. Hay muchas métricas a utilizar en el testing de un proyecto.
    - **Productividad**: cantidad de condiciones construidas por unidad de tiempo. Todo el set de condiciones y escenarios para probar. Difícil de medir productividad en los test.
        - **Productividad Diseño**: cantidad de condiciones construidas por unidad de tiempo.
        - **Productividad Ejecución**: cantidad de casos ejecutados por unidad de tiempo.
    - **Eficacia**: cantidad de incidentes que encuentro dentro del código que se convierten en fallas.
        - **Eficacia** (Pre-Release)
        - **Eficacia** (Post-Release)
    - **Calidad del Desarrollo y Release Readiness**

Unidad N°6: “Testing Automation”

- Famosa “Silver-Bullet”. Lleva tiempo, esfuerzo y mantenimiento.
- Pruebas automáticas tienen que ser un complemento de las manuales.
    - No perseguir eliminar al testing manual.
    - No buscar suplantar a los testers.
    - La automatización no debe ser una meta, si no una alternativa. No es un fin en si mismo-
- Abarcan un amplio espectro del proceso de testing.

Pirámide de Cohn

- Cuando quiero automatizar algo con mucha interfaz, necesita mucho esfuerzo ya que es muy cambiante.
- Cuando la interfaz tiene poco cambio, y UT tiene muchas cosas que probar a bajo nivel, paga más, es más útil.
- **Diagrama**:



Conclusiones

- Acumular experiencia en testing manual para entender el caso de uso.
- Setear expectativas y no esperar resultados de manera inmediata. No se puede automatizar todo.
- No perder de vista que las condiciones y casos son el activo.
- **Cuando pasa el tiempo y no se mantiene la automatización, me va a dar muchas fallas a la hora de realizar el testing.**
- **Automatizar es un proyecto y aplicación más. Tener en cuenta todos los atributos de calidad.**

Paper: “Testing Without a Map”

- **Parte del concepto de que tengo que probar y no se de dónde agarrarme. No hay documentación de la solución. No tengo “mapa”.**
- **Se puede probar algo basándonos en heurísticas. Son puntapiés que no me garantizan éxito.**
- **En muchos contextos no es necesario planear todo el testing. Con meterme a hacer exploración basándome en las heurísticas, puedo encontrar bugs y definir si el software es útil o no. Balance entre tiempo, recursos y resultados esperados.**
- Usando estos puntos podemos expresar por qué pensamos que algo es un bug.
- **HICCUP**, el producto (desde el punto de vista del tester) debe tener consistencia en su:
    - **History**: ¿Algo cambió sin aviso?
    - **Image**: ¿Se ve profesional?
    - **Comparable Product**: ¿El producto similar tiene este comportamiento?
    - **Claims**: ¿Coincide con lo que se habla sobre producto?
    - **User Expectation**: ¿El usuario espera este comportamiento?
    - **Product**: ¿Hay cambios en la terminología / cómo funciona / look and feel dentro del mismo producto?
    - **Purpose**: ¿Es consistente con el propósito del software en sí? Por ejemplo: valores negativos en el tamaño de letra en Word