Resumen 2do Parcial

SCM - Software Config Management

Item de Configuración - IC

Para cada item se conoce información sobre su configuración (nombre, version, fecha de creación, autor, etc)

Cosas de nuestro sistema que vamos a querer controlar y gestionar

- Codigo
- Documentación
- Scripts
- Bases de datos
- Modelos
- Servers
- Especificaciones de usuario

CONDICIONES PARA QUE SEA IC

- Componente principal
- Que sea utilizado por mas de una persona
- Componentes que se relacionan con otros componentes
- Que cambien en el tiempo

Configuración

Conjunto de todos los componentes fuentes que son compilados en un ejecutable consistente.

Todos los componentes, documentos e información de su estructura que definen una version determinada del producto a entregar

💡 Lo primero q hay q averiguar son los items de configuración

Línea Base:

Conjunto de ítems de configuración que forman/generan una configuración a partir de la cual puedo tomar decisiones y puedo establecer un determinado acuerdo en algún momento.

Ponerse de acuerdo que esta es la versión que funciona y qué tiene por componentes. Estos componentes me son significativos por algún motivo y me generan una configuración que me sirve de referencia. Ej: configuración inicial.

Gestion de configuración de software SCM

Disciplina orientada a administrar la evolución de Productos, procesos y ambientes.

Busca establecer y mantener la integridad de los productos del proyecto de software a lo largo del ciclo d vida del mismo

Involucra para una configuración:

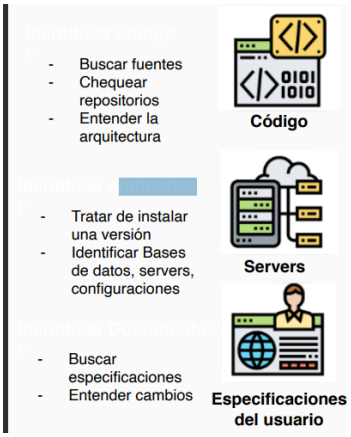
- Identificarla en un momento dado
- Controlar cambios sistemáticamente
- Mantener su integridad y origen

Acompaña la actividad de cambio con actividades de control

📌 No todo lo que involucre al proyecto va a estar bajo nuestro plan de SCM

Pasos a Seguir según SWEBOK

HAY QUE AVERIGUAR CUALES SON LOS ITEMS DE CONFIGURACION



1. **Identificar la configuración**

   Definir qué elementos (ICs) van a ser controlados por la Gestion de configuración. Es necesario contar/definir guías para que elementos son parte o no.

   Define momentos o condiciones para establecer una línea de base, o para liberarla (ósea llevarla a otro ambiente)

Clasifico los IC según proceso o producto

Proceso → se gestionan a lo largo del proyecto o ciclo. Tienen una vida util de gestion que equivale a una unidad fija de tiempo, una vez qe termina no los gestiono mas. **Nacen y mueren**. (riesgos, plan de calidad, formularios de aceptacion)

Producto → Definen al producto de SW en si. Trascienden el tiempo del proyecto. Ejemplo -. requerimientos, planes de instalación/despliegue, manual de usuario, codigo fuente

> 📌 No todas las versiones del software son líneas base

1. **Control de la configuración - asegurar mantener la integridad**
   - La identificación del propósito del cambio
   - La evaluación del impacto y aprobación del cambio
   - La planificación de la incorporación del cambio
   - El control de la implementación del cambio y su verificación

     Busco establecer un orden para hacer los cambios en el software una vez que identifique la configuración.

     Deberíamos poder trazar o trackear todos los cambios de nuestro flujo.

     Jira → Code / SQL → Testing - QA → Staging → Prod por ej es un flujo básico

📌 **Change Management Flow** → el flujo de solicitud de cambio a una configuration

SCCB - Software Configuration Control Board: Grupo de personas encargados de aprobar o rechazar cambios. Lideres. Producto. CEO, CTO. Legales. Seguridad informática.

1. **Status & Accounting de la configuración**

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

   Siempre se implementa con una herramienta tipo Git

   - Listar los ICs aprobados
   - Mostrar el estado de los cambios que fueron aprobados
   - Reportar trazabilidad de todos los cambios efectuados al baseline

   Debo poder contestar, **que cambios se realizaron al sistema? cuando? quien lo cambio? que cambio? alcance del cambio? quien lo aprobó? quien lo solicito?**

2. **Administración de Distribución y Despliegue del Sw - construcción y distribución**

   Aseguro la construcción exitosa del paquete de software, basada en los ICs requeridos parala funcionalidad a entregar para después liberarlo en forma controlada a otros entornos ya sea de pruebas, producción, usuario final, etc.

   Desplegar → sabarlo del ambiente en el que se desarrollo

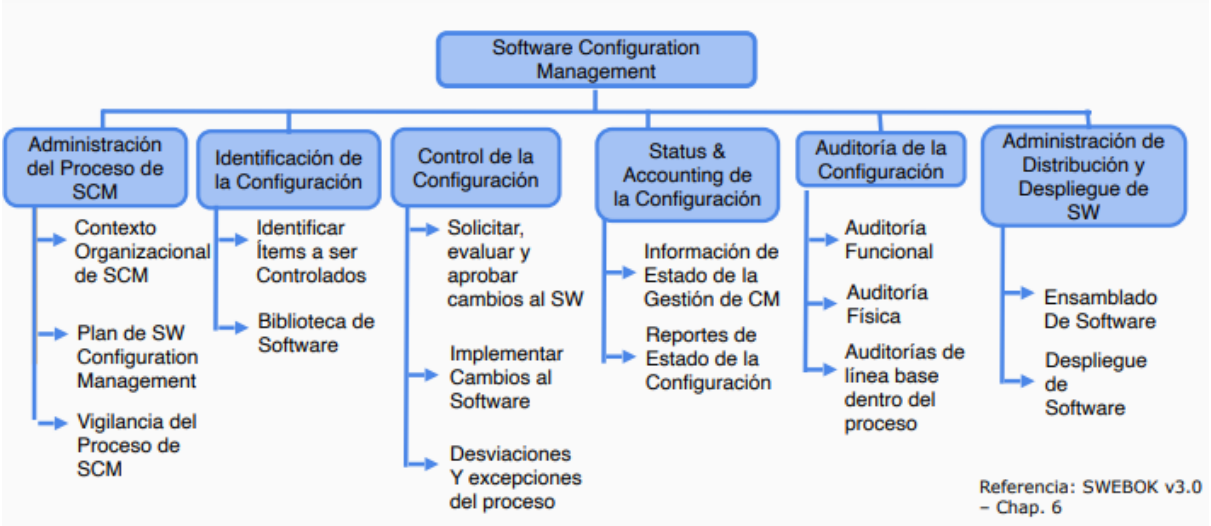
   Se divide en 2 partes:

   1. **Software Building** → combinar versiones correctas de ICs, usando la configuración adecuada en un programa ejecutable
   2. **Release Management** → la administración, identificación y distribución de los elementos de un producto (ej: programa ejecutable, documentación. notas de release, datos de configuración)

   Además del ejecutable/paquete de sw, **esto comprende la administración, identificación y distribución de un producto**

// tenemos el código y hicimos los cambios pedidos. Debemos construir nuestra configuración y habilitarla para que pueda ser usada por los usuarios

SCM según SWEBOK



Auditoria de la configuración

Busca hacer una verificación del estado de la configuración para determinar si se cumplieron los requerimientos especificados.

Se puede realizar con diferentes niveles de formalidad -. Revisiones informales basadas en checklists, o pruebas exhaustivas COPUAR

Tipos de auditoria

- **Funcional** → Que haga lo que dice que hace
- **Física** → verifica la configuración del producto en cuanto a la estructura especificada. (que este en el repo que tiene q estar, etc)
- **De Proceso →** valida y verifica el cumplimiento del proceso SCM

Son un control Posterior

💡 Ejemplo → testing para funcionalidad. Otro de funcionalidad es por ej el componente critico de contabilidad de un banco, que permite hacer un control más exhaustivo sobre el código comparando el código viejo vs el nuevo para comprobar que las modificaciones estén bien.
Física → Que no haya componentes extras
De proceso → Las empresas suelen tener un area que se dedica a fijarse que se sigan todos los pasos para desarrollar

Definiciones importantes

Branch

Crear líneas de desarrollo separadas, que usan la línea base del repositorio como punto de partida.

Permite a los miembros del equipo trabajar en multiples versiones de un producto usando **el mismo set de items de configuración**

Linea de Base - Baseline

Representa un estado de la configuración de un conjunto de items en el ciclo de desarrollo. Puede tomarse como punto de referencia para una siguiente etapa del ciclo

Se establece porque se verifica que la configuración de los items satisface algunos requerimientos funcionales o técnicos

hay componentes q van a sufrir cambios, que no van a sufrir cambios, que van a desaparecer, o una combinación de todo.

💡 **Todas las líneas base son configuraciones, pero no todas las configuraciones son líneas base**

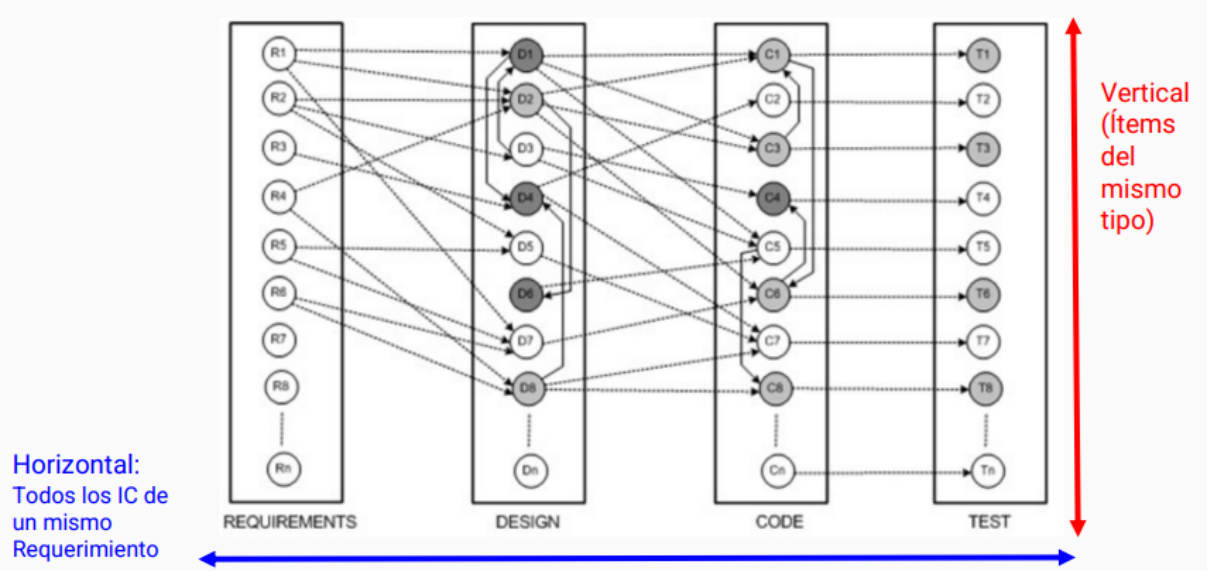
Las líneas bases para modificarlas tengo q lograr un acuerdo

Trazabilidad en los Items de Config

Trazabilidad horizontal → cuando llevamos un IC o doc desde el momento que origino hasta el momento que se eimplemento en el producto SW. Poder trazar

su origen

Trazavilidad vertical → Todos los IC son del mismo tipo, por ej testing, code, requirenments, etc



Como aplico SCM a la vida real

Un plan SCM define principalmente:

- Lista items de configuración (IC) y en que momento ingresa al sistema
- Definir estándares de nombres, jerarquías de directorios, estándares de versionamiento
- Definir políticas de branching y de merging. Políticas de entregables
- Definir los procedimientos para crear builds y releases.
- Definir reglas de uso de la herramienta de CM y el rol del administrador de la configuración

Definición y administración del proceso SCM

La realización del Plan SCM y sus etapas:

1. Identificación de la configuración
2. Procesos de Control de Cambios
3. Auditorias de software
4. Release management

5. Políticas de manejo de branches

Es la etapa previa al arranque el proyecto. Se debe comprender el contexto organizacional

> 💡 preguntas a hacerse: existe ya un proceso SCM en la org? ¿que establece? Que herramientas hay disponibles? Que cosas deberemos incluir o cambiar en el plan SCM?

Se define:

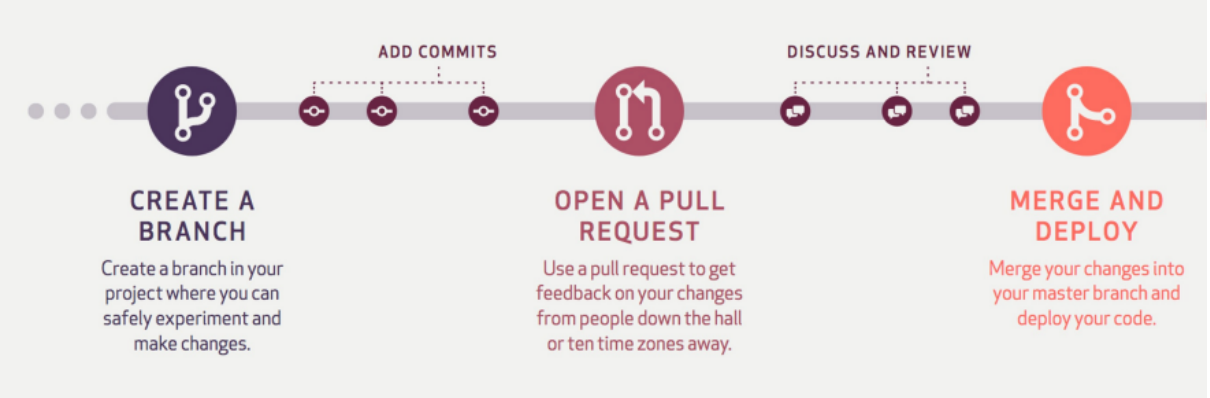
- Lista de ICs y en que momento ingresan al CM
- Estándares de nombres, jerarquías de directorios y estándares de versionamiento
- Políticas de branching y de merging
- Procedimientos para crear builds y releases.
- Reglas de uso de la herramienta de CM y el rol del administrador de la configuración
- Contenido de los reportes de auditoria y los momentos en que estas se ejecutan

Todo esto puede definirse a nivel organizacional o proyecto

Condiciones para que el proceso SCM este Definido

Se puede decir que el proceso esta definido si:

- Esta definido como se registran los cambios de software (JIRA< TFS)
- Se establecieron los roles que aprueban o rechazan los cambios
- Esan identificados los repositorios de los artefactos
- Hay establecida una política de trabajo sobre los repositorios
- Están identificados los momentos en los que se realizan las líneas bases (gitflow)
- Los cambios son gestionados completamente, desde su ingreso hasta su puesta en producción



Software Building

Local Build → El developer lo hace localmente en su entorno, corre pruebas unitarias (UT)

Integration Build → Busca generar el entorno completo para pruebas e integración

Nightly Build → Busca ejecutar la construcción en forma diaria y generar reportes con información sobre estabilidad, tiempo de build, etc

Release Build → Se disparan cuando un administrador decide crear una nueva version a liberar, o por el sistema de integración si se usa el modo **deployment continuo.**

Los builds:

- Deben ser automáticos
- Deben permitir la generación de reportes (cada build genera un reporte del estado del mismo)
- Reducen la cantidad de defectos
- Mejora la reproducción de problemas y trazabilidad
- Mejora la performance del equipo de desarrollo

Build & Deployment Pipelines

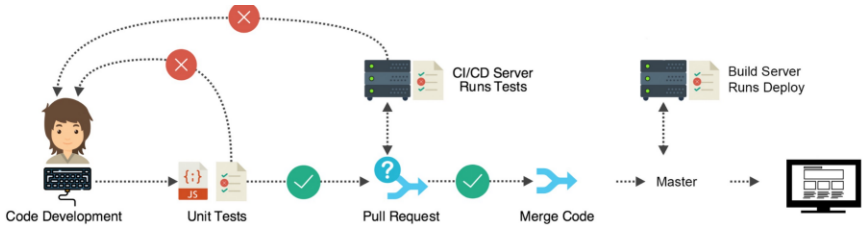
La manifestación automatizada del proceso para llevar software desde la aprobación del cambio de SCM hasta las manos del usuario

Asociado al deployment

Incluye el camino desde que se compila y construye el software, seguido por el proceso de varias etapas de testing y deployment. Requiere colaboración entre

individuos y equipos.

Netamente la parte de construcción del producto software, no incluye lo que sucedió antes con el requerimiento, cómo llegó y demás



Release Management

Asegurar la construccion exitosa del paquete de software basado en los ICs requeridos y liberarlo en forma controlada a otros entornos.

Un release es un ejemplo de una potencial linea base

Deployment

Una vez generada la version o release debemos encontrar el mecanismo mas efectivo para hacer el deploy controlado a los usuarios

Se debe evaluar:

- Que usuarios deben recibir los cambios
- En qué forma hago el despliegue
  - Entornos por los que debe pasar
  - Procesos de roll forward
  - Procesos de roll back
  - Validación de despliegue correcto-incorrecto
- Riesgos de despliegue y como minimizarlos
- Aprobación por negocio o area de QA
- Quienes harán el deployment de la version definida

conceptos de despliegue → tener en cuenta el punto de no retorno, cuando ya no puedo hacer rollback. Todo eso lo ve el software configuration control board

Definiciones

Version → instancia de un sistema que es funcionalmente distinta en algún aspecto de otras instancias

Variante → instancia de un sistema que es funcionalmente igual a otras instancias, pero difiere a nivel no funcional (version Linux vs version windows)

Release → la distribución del software fuera del entorno de desarrollo.

Basic Deployment

Todos los nodos de la aplicación son visualizados en el mismo momento con la nueva version.

Ventajas → Simple de usar, no hay mecanismos específicos que deba realizar para este deploy

Desventajas → Alto riesgo de downtime

Blue green deployment

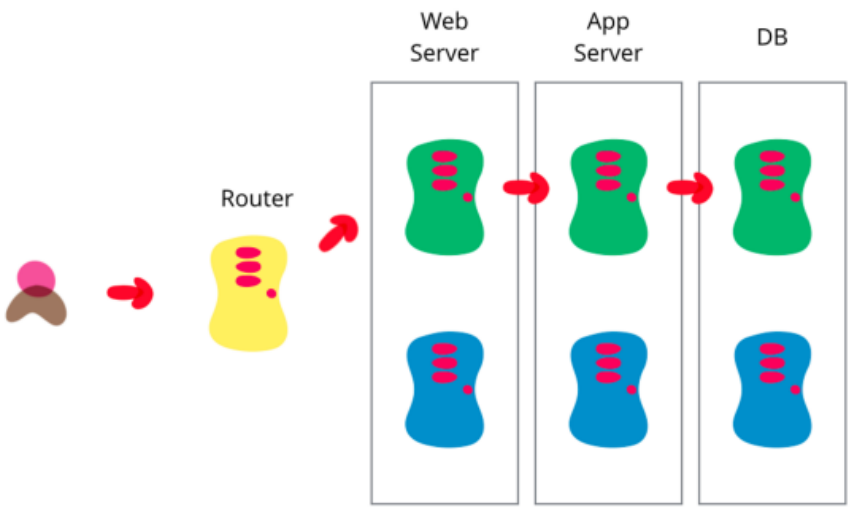
Dos ambientes lo mas similares posibles, uno blue (continee la version anterior) y el otro green (va a contener la nueva versioin).

El despliegue se realiza sobre uno de los ambientes, mientras el otro contiene la version anterior. Una vez que el despliegue se hizo en green, blue queda en standby con lo anterior en caso de necesitar hacer un rollback

Un balanceador selecciona que grupo de servers se usa como productivo

Permite actualizar una aplicación o sistema mientras se está usando, sin necesidad a esperar un horario de poco uso.

Desventajas → en organizaciones grandes puede terminar siendo muy costosa



Canary Deployment

Consiste en desplegar el cambio en un set controlado de nodos (usuarios) y ir chequeando el comportamiento. Progressive rollout of an app, splits traffic between the already deployed version and the new version, controlling how it works.

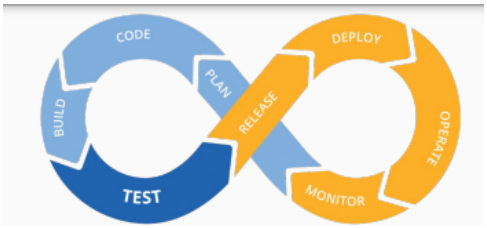
Es de las formas mas comunes de deployear

Ventajas → El despliegue es en fases, progresivamente en los nodos. En caso de fallar, se sacan los nodos que se habían incorporado. **Solo existe 1 ambiente de producción**

Desventajas → Es complejo de testear en producción, herramientas de monitoreo complejas

Continuous Integration & Continuous Delivery CI CD

> En continuos deployment todo es automático



Integración Continua CI

Realiza la integración de código al menos 1 vez por día para minimizar problemas.

Practicas:

- Repositorio de código único
- Automatizar el proceso de build
- Hacer el build testable automáticamente
- Todo commit debe construirse por una herramienta de integración, no por el dev
- Se prueba en un clon de produccion

- el build debe ser rápido
- Cualquiera debe poder obtener fácilmente la ultima version del ejecutable
- Todos deben poder ver que pasa en el proceso de integración
- Automatizar el proceso de deployment

Responsabilidades del equipo

Hacer check in de codigo frecuente

No subir código roto

No subir código no testeado

No subir código cuando el build no pasa

No irse hasta que el build central compile

Ventajas

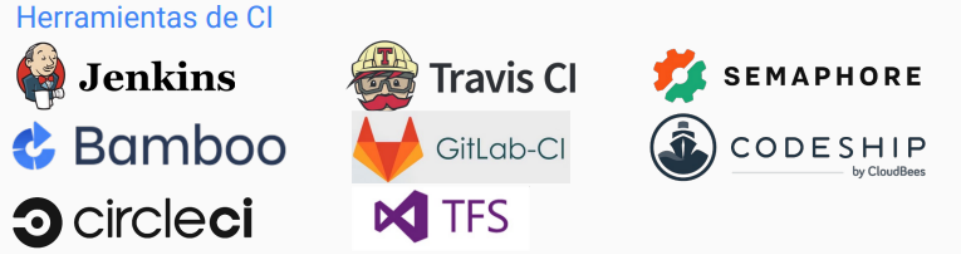
No hay integraciones de días de trabajo

Mejora la visibilidad y comunicación

Atrapa errores de integración complejos en forma temprana

Mayor rapidez para lanzar software al reducir problemas de integración

Finaliza con el “en mi maquina funciona”

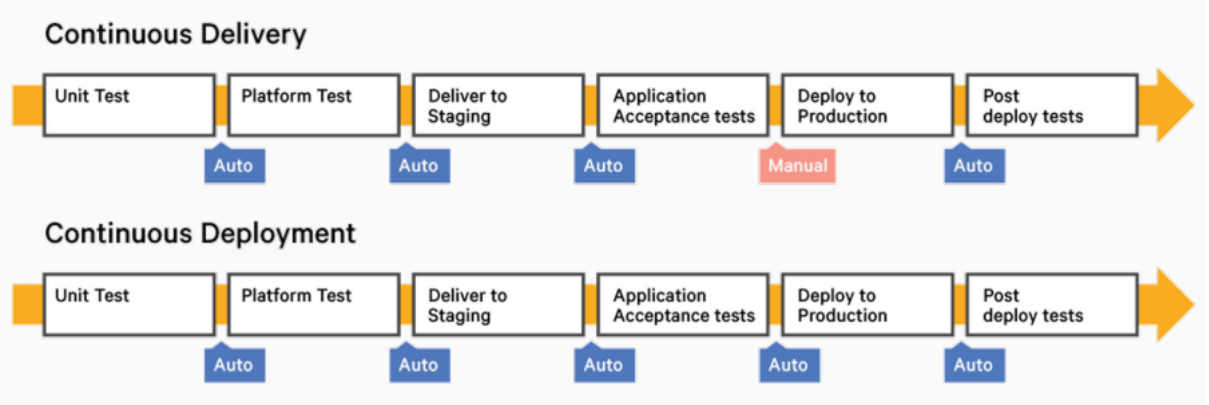


Continuous Delivery

Asociado a CI. **Conjunto de prácticas que permiten asegurar que el software pueda ser desplegado en producción rápidamente y en cualquier momento**

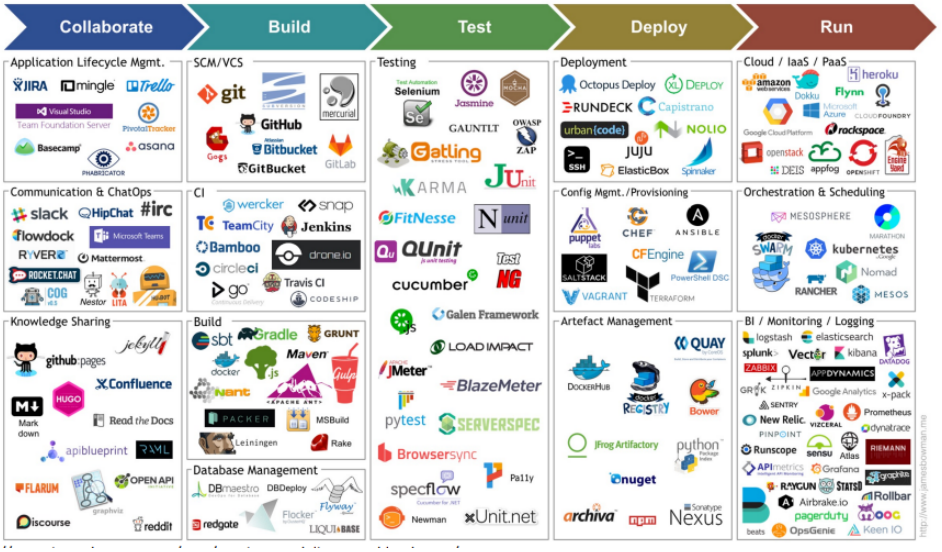
Hace que cada cambio llegue a un entorno de **staging** o semi productivo, donde se corren pruebas del sistema

> Todo es automático salvo LE PASO A PRODUCCION QUE ES MANUAL



CONTINUOUS DEPLOYMENT → paso siguiente al continuous delivery, donde el despliegue a producción también es automático

Landscape de herramientas



Dev Ops

El que desarrolla no escribe los requerimientos, ni hace el testing funcional, ni pasa a la producción por lo que los desarrolladores no consideran que los entornos pueden tener diferencias, que la puesta en producción es compleja, la escalabilidad, performance, disponibilidad y tolerancia a fallos no se tenían en cuenta, etc.

Faltaba comunicación y colaboración

Ante un nuevo cambio, los devs lo hacían pero ops solo ve los posibles nuevos riesgos

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

Practicas de DevOps

Planificación del cambio → conversación y colaboración

Coding & Building → automated build pipelines, infrastructure as code

Testing → Automated testing, chaos testing/fault injections

Release & Deployment → Continuos Integration, continuous delivery

Operations & Monitoring → Virtualization & containers, monitoring & alerting tools

CALMS

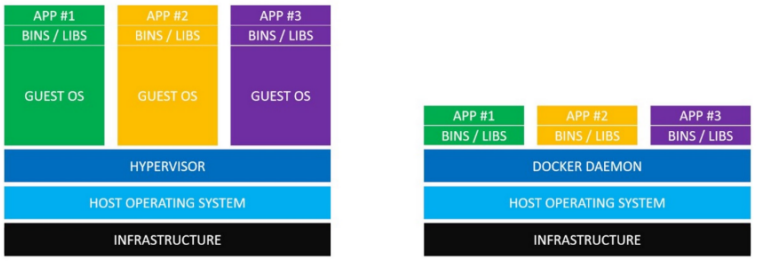
Cultura → Se dueños del cambio para mejorar la colab y comu

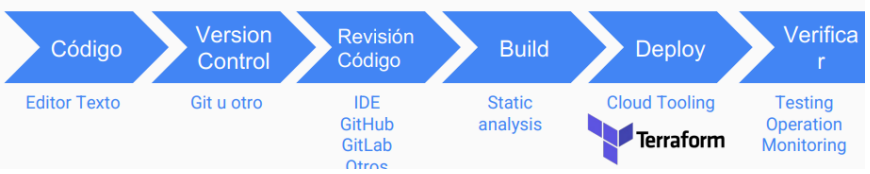
Automatización → Eliminar el trabajo manual y repetitivo lleva a procesos repetibles y sistemas confiables. Menos error humano

Lean → Remover la burocracia para tener ciclos mas cortos y menos desperdicio

Metricas → usar datos para refinar los ciclos

Sharing → Compartir experiencias de éxito y falla para aprender de eso





But i only changed one line of code - Lectura

This is what is referred to as a configuration. The ability to properly distinguish the appropriate parts that must be used to build each type of engine is critical to assure consistent, safe production and use of aircraft engines.

Software systems are made of parts that cannot be touched, picked up, physically put in place, or manipulated. If you lose track of one of the software pieces, you have to re-create it.

Software Configuration Management (SCM) has been defined as the art of identifying, organizing, and controlling modifications to software [1]. It should be performed across the entire software development and maintenance life cycle.

Configuration Identification

Identificar las critical parts that must be managed.

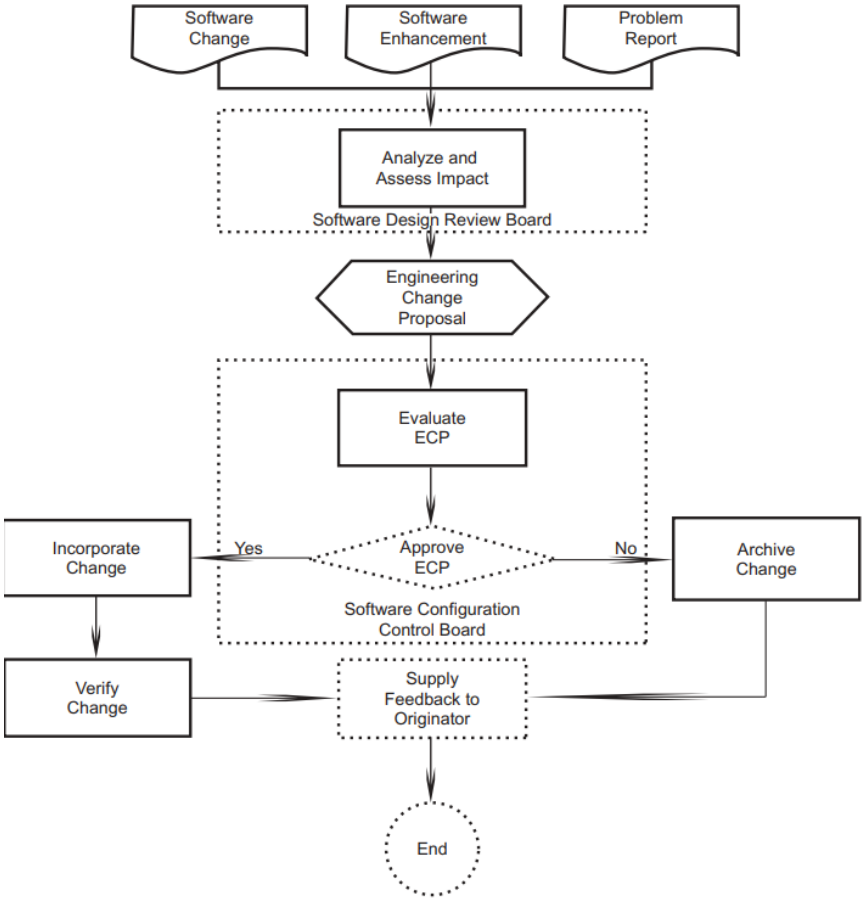
CIs are the objects required to design, develop, build, maintain, test, and field a software product.

Items that are not identified cannot be managed. If you need it to accomplish your development task, then you probably should assign an identification number to it

Configuration Control

The process of controlling and limiting changes made to software assets. Making sure that changes to software assets are only allowed to occur after they have been analyzed, evaluated, reviewed, and approved by a group (Configuration Control Board CCB) authorized to control changes to software assets.

The CCB should include representation from program management, systems engineering, software engineering, software quality assurance, software configuration management, independent test, and a customer representative



Configuration Status Accounting

SCI must be accounted for.

It includes the tracking of changes to SCI's, and provides the ability to determine the status of each item at any phase of the software development or maintenance process.

Status accounting involves gathering information to answer the following questions:

- 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?

ACA ES DONDE ENTRAN LOS REPOSITORIOS. Changes to the status of the assets must be documented and tracked.

List of sample reports that the status accounting function should provide.
• Transaction log.
• Change log.
• SCI delta report.
• Resource usage report.
• SCI status report.
• Changes in progress report.
• Change request status report.
• Change completion report by developer, application, etc.

Configuration Auditing

People are more likely to follow the approved practices if they know someone is checking to see that they do.

Audits should be conducted to help assure that software development policies, processes, and procedures are being consistently followed and adhered to.

A software configuration audit should be periodically performed to ensure that the SCM practices and procedures are rigorously followed. The integrity of the software baseline must be assessed. It is recommended that a software configuration audit be performed before every major baseline change.

A **functional configuration audit** is intended to verify that the software functions as defined by the software requirements documentation. A **physical configuration audit** is intended to verify that all the items identified to be included in software release are actually included and that no additional items are included.

Testing

Conceptos generales

Actividad para evaluar si los resultados obtenidos en la ejecución del software coinciden los los resultados esperados, con el propósito de determinar si el sistema está libre de defectos ajo condiciones especificas. Involucra la ejecución de un componente de software o de un sistema para evaluar una o varias propiedades de interés

Definición por Profesor Cem Kaner: **Es una investigación técnica empírica ejecutada para proveer a los stakeholders con información de la calidad del producto o servicio que se encuentra a prueba.**

💡 Las pruebas por si solas encuentran fallas, no defectos. (solo pueden demostrar su presencia, no su ausencia)

Partimos de la suposición que el SW a probar *tiene defectos*, y buscamos tantos como podamos

Adoptar la actitud de que el Sw no tiene errores nos inclina hacia ello

Importancia del testing

Los bugs pueden ser costosos o incluso peligrosos. Los bugs de software pueden causar potencialmente problemas económicos o de pérdidas humanas.

Cada vez existe software más crítico y más complejo.

El Software está escrito por personas, y las personas cometen errores entonces el software puede cometer defectos que llevarán potencialmente a falla

Crisis del Software

- ¿Por qué lleva tanto tiempo terminar los programas?
- ¿Por qué es tan elevado el coste?
- ¿Por qué no podemos encontrar todos los errores antes de entregar el software a nuestros clientes?
- ¿Por qué es tan difícil constatar el progreso durante el desarrollo?
- ¿Por qué es tan difícil calcular cuánto tiempo va a costar?

Se dice que el Sw esta en crisis porque pasaba antes y pasa ahora. Ante esto nace la Ingeniería de software

Objetivo del testing

Encontrar fallas en el producto de forma eficiente (lo más rápido y barato posible) y de forma eficaz (encontrar la mayor cantidad de fallas, no detectar fallas que en realidad no lo son, encontrar las más importantes)

Porque se producen fallas

- Requerimientos poco claros

- Requerimientos incorrectos
- Requerimientos cambiantes
- Diseño Complejo
- Desarrolladores sin experiencia
- Lógica Complicada
- Falta de tiempo
- Falta de conocimiento del producto
- Testers sin experiencia
- Problemas del entorno o de sistemas

Calidad → además de lo visto antes, que el Sw no tenga fallas

QA - Quality Assurance

Patrón planificado y sistemático de todas las acciones necesarias para brindar una la confianza de que un producto o componente cumple con los requerimientos técnicos establecidos

La prueba se software es una actividad realizada dentro de QA

QUALITY ASSURANCE SE HACE A LO LARGO DE TODO EL PROYECTO, DESDE QUE SE ENVISIONA HASTA SU FINAL. tESTING ES UNA ETAPA QUE ES PARTE DEL QA

Asegurar vs Controlar la calidad

SQA (SW QA)vs testing

testing esta adentro de SQA. Se prueba si tiene calidad al final de haber realizado todos los testeos

Una vez definidos los requerimientos de calidad tengo que tener en cuenta que la calidad no puede inyectarse al final, y la calidad de un producto depende de las tareas realizadas durante el proceso

SQA detecta errores en forma temprana, ahorrando esfuerzos, tiempo y recursos

💡 Una prueba es exitosa si encuentra fallas

Ciclo de testing

1. Miro los requerimientos
2. Establezco las condiciones de prueba y despues los casos
3. Chequeo si llego al resultado esperado

Lo pongo en practica:

1. Agarro el componente
2. Armo el entorno de prueba
3. Ejecuto los casos
4. Reviso el resultado obtenido
5. TOMO NOTA DE TODO

Equivocación → acción humana que produce un resultado incorrecto

Defecto → Paso, proceso o definición de dato incorrecto. Ausencia de cierta característica

Falla → Resultado de ejecución incorrecta

Incidente

Toda ocurrencia de un evento que ocurre durante la ejecución de una prueba de software que *requiera investigación*.

Puede deberse a

- Un defecto en el SW
- Un defecto en los casos de prueba
- Un defecto en el ambiente

NO TODO INCIDENTE ES UNA FALLA

Depuración

Eliminar un defecto del SW

No es una tarea de prueba, es una consecuencia

> La depuración puede ser fuente de introducción de nuevos defectos

Proceso:

1. **Detectar** → dada la falla debemos hallar el defecto
2. **Depurar** → eliminar el defecto, encontrar su razón, su solución y aplicarla
3. **Volver a probar** → asegurar que sacamos el defecto y que no introducimos otros
4. **Aprender para el futuro**

Hasta cuando debo probar?

Probar es el proceso de establecer confianza en que un programa hace lo que se supone que tiene que hacer

Como nunca voy a poder demostrar que un programa es correcto, continuar probando es una decision económica

Estrategias para definir cuando ceso las pruebas:

- Pasa exitosamente el conjunto de pruebas diseñado
- “Good Enough”: cierta cantidad de fallas no críticas es aceptable (umbral de fallas no críticas por unidad de testing)
- Cantidad de fallas detectadas es similar a la cantidad de fallas estimada

Abaratar el testing

Es un proceso caro y trabajoso

La forma de abaratarla sin degradar su utilidad es diseñando el software para qque sea testeado.

○ Diseño Modular
○ Ocultamiento de Información
○ Uso de Puntos de Control
○ Programación NO egoísta

Enfoques de prueba

Caja negra

Prueba funcional, producida por los datos o por la entrada/salida

Prueba lo que el software debería hacer, basándose en la definición del modulo a probar. **nos desentendemos del comportamiento y estructura interna del componente**

Como se hace:

- Selecciono subconjuntos de datos de entrada posibles, esperando que cubran un conjunto extenso de datos de otros casos de prueba posibles
- Supongo que la prueba de un valor representativo de cada clase es equivalente a la prueba de cualquier otro valor

  Cada subconjunto seria la clase de equivalencia

Estas pruebas se llaman **Pruebas por partición de Equivalencia**

Criterio de selección caja negra

- Variaciones de Evento
- Clase de Equivalencia
  - De entrada
  - De salida
- Condiciones de Borde
  - Ingreso de valores de otro tipo
  - Integridad del Modelo de dato
- De dominio
- De entidad
- De relación

Condición de prueba

Descripciones de situaciones que quieren probarse ante las que el sistema debe responder.

Casos de prueba

Lotes de datos necesarios para que se dé una determinada condición de prueba

Criterio de selección de casos

Condición para seleccionar un conjunto de casos de prueba

De todas las combinaciones posibles solo seleccionaremos algunas.

> La menor cantidad de ellas que tengan mayor probabilidad de encontrar un defecto no encontrado por otra prueba

Partición de los casos

Todos los posibles casos de prueba los dividimos en clases
Todos los casos de una clase son equivalentes entre sí:

Detectan los mismos defectos

Con solo ejemplos de cada clase cubrimos todas las pruebas

> El éxito está en la selección de la partición !!!

Partición de Equivalencia - Como se hace

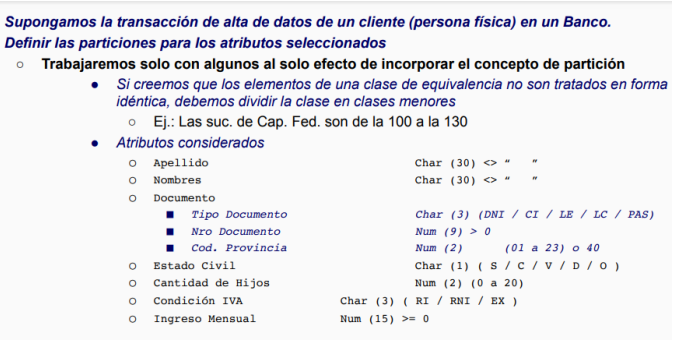
1. Identificar las clases de equivalencia
2. Definir casos de prueba

Para definir las clases de equivalencia, las divido entre Clases Validas y Clases Invalidas

ejemplo → si tengo un rango de valores, elijo 1 valida y dos invalidas

Si creo que los elementos de una clase de equivalencia no son tratados en forma idéntica, debo dividir la clase en clases menores

EJEMPO



💡 Los casos de prueba que exploran las condiciones de borde producen mejores resultados que las que no lo hacen

Condición de borde

En rangos dé valores, enfocarnos en valores internos y externos al rango cerca de los extremos

Lo mismo para los datos de salida

Prestar atención especial a los archivos/tablas vacíos, primer registro/fila, ultimo registro/fila, fin del archivo/fila.

Clases invalidas

Se deben probar además de valores inválidos del mismo tipo de dato, valores de otros tipos de datos.

A veces estas validaciones ya las hace el entorno de desarrollo

> Muchas veces las combinaciones de datos validos e inválidos producen clases Validas-Invalidas

Las condiciones cruzadas deben ser tenidas en cuenta

Conjetura de errores - Prueba de Sospechas

1. Enumero una lista de errores posibles o de situaciones propensas a tener errores
2. creo los casos de prueba basados en estas situaciones

El programador es quien nos puede dar la información más relevante, y es un proceso muy efectivo

La creatividad juega un papel clave

Cuando hacer conjetura de errores?

- Cuando hubo un componente o parte hecho a las apuradas
- Un componente modificado por varios personas en distintos momentos
- Un componente con estructuras anidadas, condiciones compuestas, etc

- Un componente que sospechamos fue armado por copy paste de varios otros componentes

Enfoque Caja Blanca - open box

Prueba estructural del software

Método en el cual la estructura interna, el diseño y la implementación del software es conocido por el tester.

> The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system.

El tester debe conocer el lenguaje de programación

Se usa para incrementar el grado de cobertura lógica interna del componente tambien

Grados de cobertura

Cobertura de sentencias

prueba con instrucción

Cobertura de Decisiones

Prueba con salida de un if o while

ejemplo calculando raíz cuadrada, pruebo con 4 y con -10

Cobertura de Condiciones

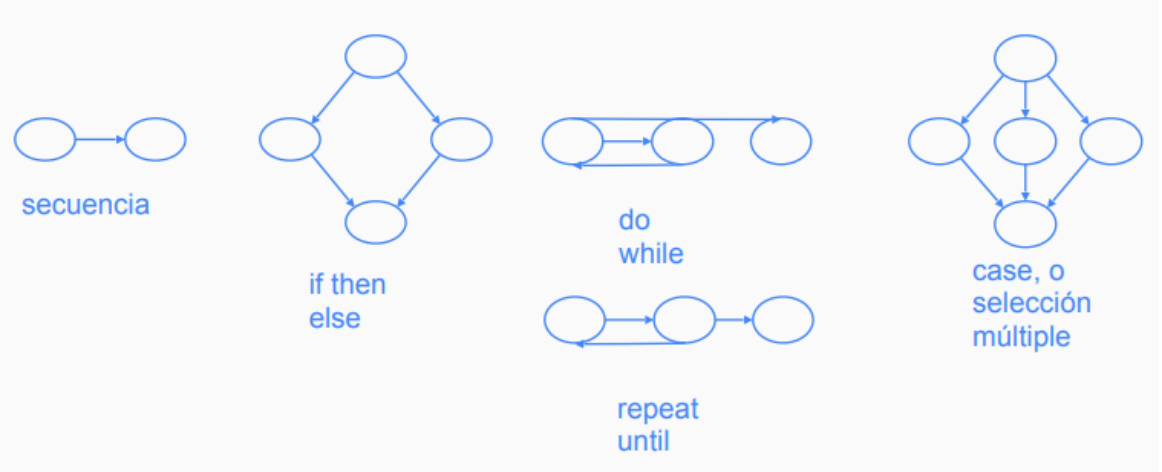
Prueba cada expresión lógica de los if y while

Prueba del Camino Básico

Prueba todos los caminos independientes

> 💡 **Camino independiente** → un nuevo conjunto de sentencias de procesamiento o una nueva condición

Se representa el flujo de control de una pieza de codigo a través de un grafo de flujo



Complejidad Ciclomatica

Métrica del software que proporciona una medición cuantitativa de la complejidad lógica de un programa

- Cantidad de caminos independientes

V(g) = A - n + 2 → N = nodos y A = aristas

> ⚠ Ninguna técnica de generación de condiciones y casos es completa, se recomienda combinarlas para complementar sus ventajas.
> Muy importante la conjetura de errores

Prueba de Caja Gris

Conoces los requerimientos (caja negra) y una parte de la implementación (caja blanca) que te permite generar condiciones adicionales que nunca podrías poner solo con caja negra.

Prueba que combina elementos de la caja negra y caja blanca.. **Conocimiento parcial no total**

No es 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.

Casos de prueba

Prueba Unitaria

Se realiza sobre una unidad de código claramente definida

Generalmente lo realiza el area que construyó el modulo

Se basa en el diseño detallado.

Comienza una vez codificado, compilado y revisado el modulo

Los módulos altamente cohesivos son los mas fáciles de probar

Pruebas de integración

Orientada a verificar que las partes de un sistema que funcionan bien aisladamente tambien funcionen bien en conjunto

No incrementales → Bing Bang

Incrementales → bottom up, top-down, sandwich

sándwich → de arriba hacia abajo y de abajo hacia arriba encontrándome ene l medio

Las claves son conectar de a poco las partes mas complejas y minimizar la necesidad de programas auxiliares

Pruebas de Aceptación de usuario

Prueba hecha por los usuarios para verificar que el sistema se ajusta a sus requerimientos

Las condiciones de prueba están basadas en el doc de requerimientos

Es de caja negra.

> 💡 volumen prueba para lo que fue desarrollado el software, stress busca romper el límite del sistema. stress es caja negra.

Pruebas no funcionales

Stress, performance, seguridad, usabilidad, recuperación, portabilidad, etc.

Prueba de Stress

orientada a someter al sistema **excediendo los limites de su capacidad** de procesamiento y almacenamiento

Se tienen en cuenta situaciones no previstas originalmente

Performance:

Orientada a verificar que el sistema soporta los tiempos de respuesta definidos en la cuantificación de requerimientos en las condiciones establecidas.

Se evalúa la capacidad de respuesta con diferentes volúmenes de carga.

Ayudan a identificar “cuellos de botella” y causas de degradación

Van de la mano de las pruebas de volumen

Prueba de Volumen

Orientada a verificar que el sistema soporta los volúmenes máximos definidos en la cuantificación de requerimientos

Cap de almacenamiento y de procesamiento

Pen Testing / Ethical hacking - Seguridad

La práctica de testear un software, red o aplicación web con el fin de encontrar vulnerabilidades de seguridad que un hacker pueda explotar

El test en si puede automatizarse o ser hecho de forma manual. Lo encontrado debe reportase

---

— fin pruebas no funcionales

Prueba de Regresión

busca verificar que luego de introducido un cambio de código, la funcionalidad original no se alterara

Reuso condiciones y casos

Prueba de humo - Smoke Test

Verificar de manera rapida que ne la funcionalidad del sistema no haya fallas que interrumpan el funcionamiento basico del mismo. Se busca ver que algo que compilo, corra. **Muy alto nivel.** Rapida

Prueba Alfa y Beta

Se entrega una primera version al usuario que se considera lista para ser probada

- Suele estar plagada de defectos
- Es una forma económica de identificarlos
- Una alternativa valida es ejecutar en paralelo

Alfa → la hace el usuario en mis instalaciones

Beta → las hace el usuario en sus instalaciones

Ciclos de testing

Conocimiento

- Caja Negra
- Caja Blanca

Tipo de Ejecución

- Automatizada
- Manual
- Semi-automatizada

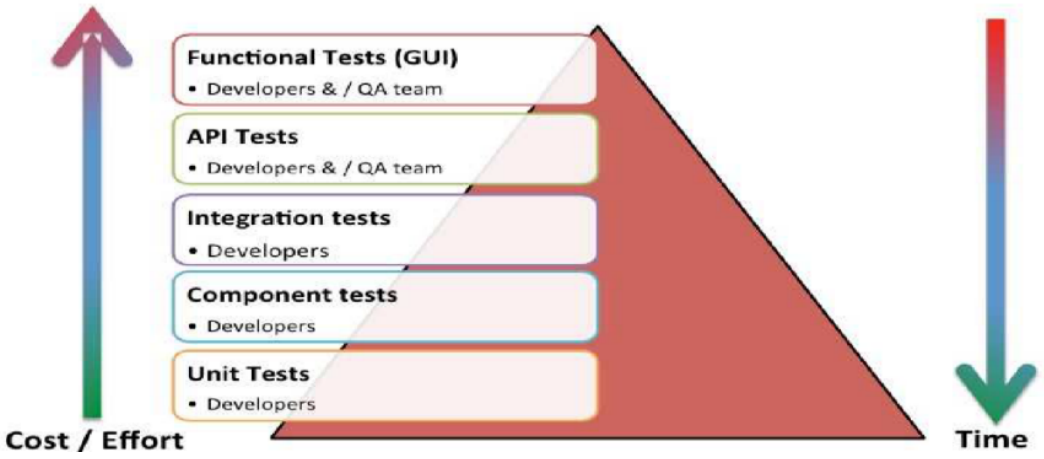
Tipo de Prueba

- Funcionales
- No funcionales

Por alcance - nivel

- Unitarias
- De integración
- De sistemas
- De integración del usuario

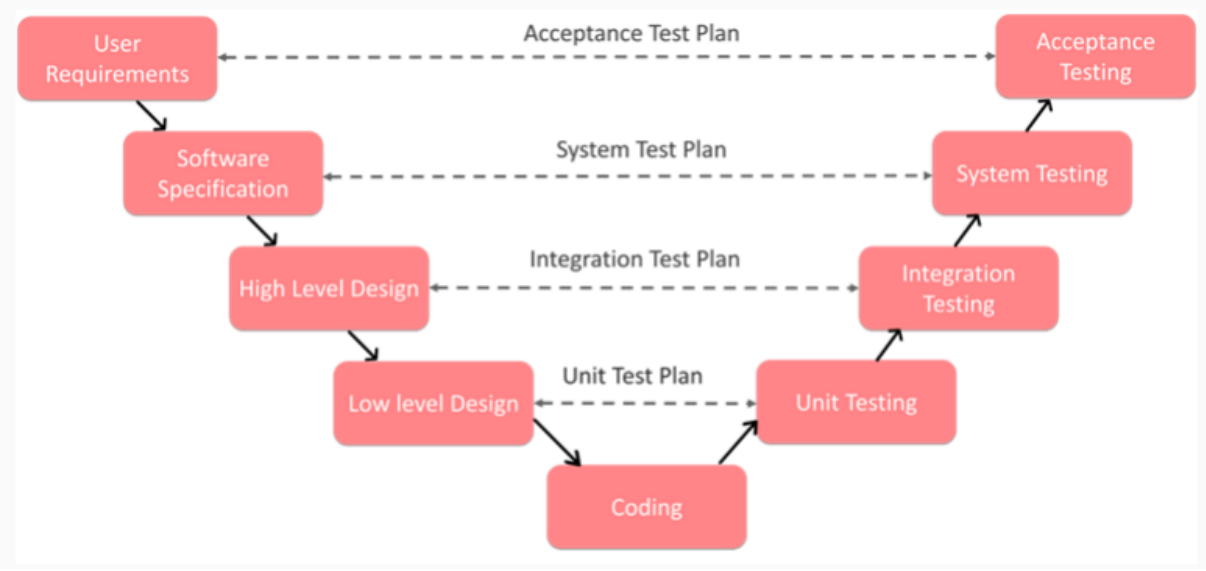
Pirámide de testing



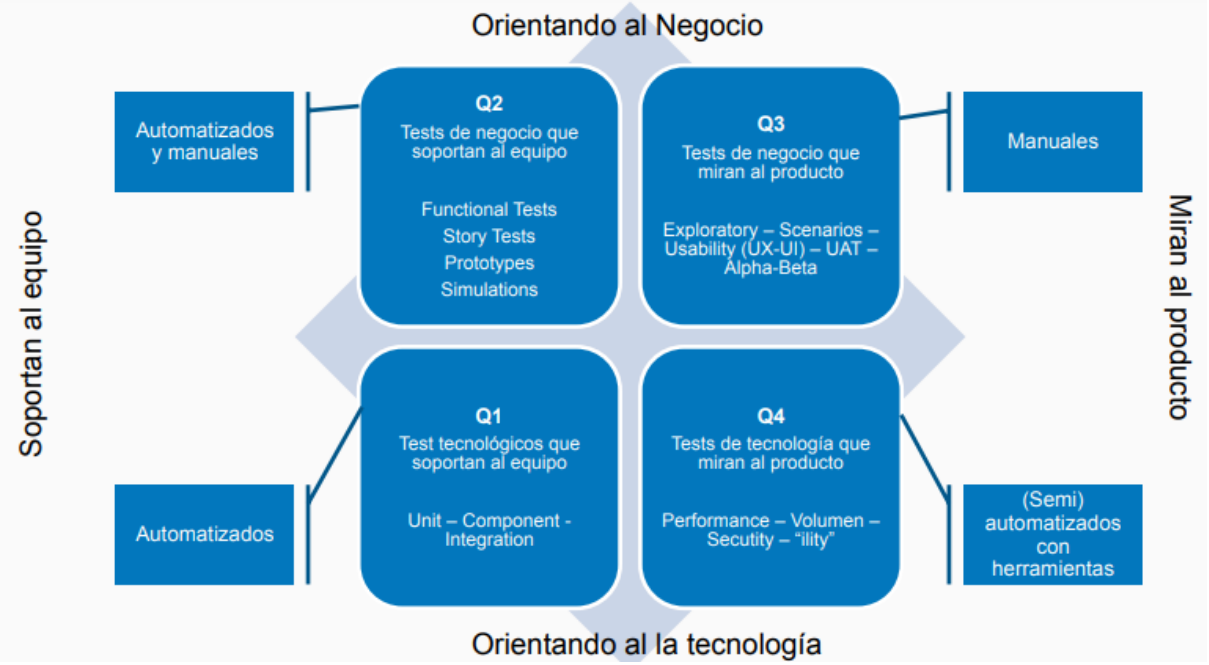
Ciclo de vida en V

Sugiere que cuando tengo definidos los requerimientos, ya puedo hacer pruebas de aceptacion, si tengo la arwuitectura puedo pensar tests de integracion, etc.

Una vez que ya estoy en la etapa, puedo ir pensando los tests para ella.



Cuadrantes, niveles y tipos de testing

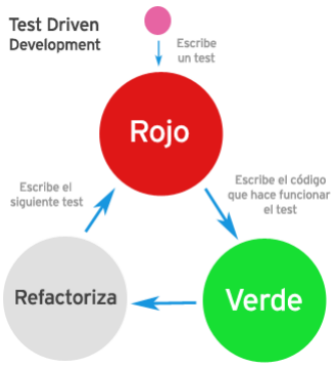


Agile testing TDD - Test Driven Development

armar los tests antes de armar el código.

Ventajas:

- Aplicaciones mas modulares, flexibles y escalables
- Codigo menos acoplado y más mantenible
- El codificador puede hacer refactoring en cualquier momento con bajo riesgo
- Soporta regression de bugs
- Mejor cobertura de codigo



Desventajas:

- Similares errores en el código y en el test case
- Algunas funcionalidades son difíciles de probar con estos tests
- Mantenimiento de los tests
- Puede llevar a un falso nivel de confianza sobre la calidad del codigo

A tener en cuenta

> La definición del resultado esperado es parte **integral y necesaria** de la prueba

Un programador debería evitar probar su propio componente

Es necesario revisar a fondo el resultado de la prueba

No suponer a priori que no se encontraran errores

Probar es un desafío intelectual

El objetivo es encontrar fallas

- Las pruebas no mejoran el SW, sólo muestran cuántas fallas se han producido debido a distintos tipos defectos
- El buen diseño y construcción no solo benefician a las pruebas, sino también a la corrección de los componentes y su mantenimiento
- El No probar No elimina los errores, ni acorta tiempos, ni abarata el proyecto
- Lo más barato para encontrar y eliminar defectos es NO introducirlos
- Hay herramientas que ayudan a automatizar las pruebas, pero requieren esfuerzo para crear y mantener y ayudan aunque no son suficientes

💡 **Prueba estática** → leer el código.

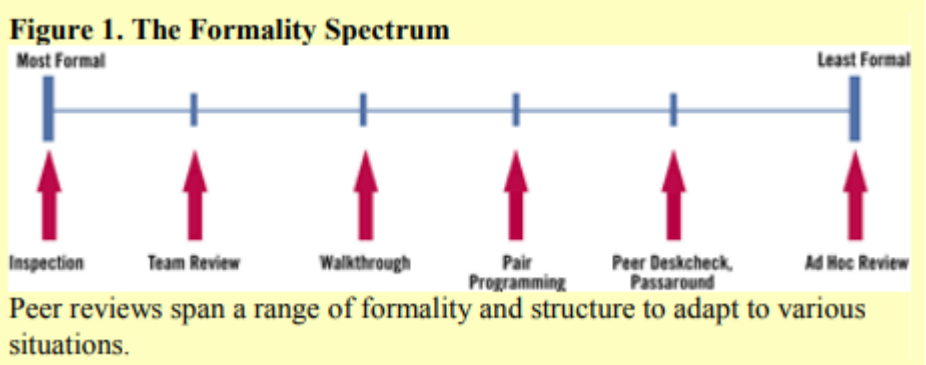
Prueba dinámica → ejecuto código. todo lo que es testing

La estática va primero. Puedo mandarlo a inspección o peer review una vez que compile, previo a una prueba de integración.

Cuando llego a testing tengo el código hecho, etc. **Una prueba estática es SQA**

When two eyes aren't enough

(todo prueba estática)



Peer review

The most formal reviews, exemplified by inspections, have several characteristics:

• Defined objectives
• Participation by a trained team
• Leadership by a trained moderator
• Specific roles and responsibilities
• A documented review procedure
• Reporting of results to management
• Explicit entry and exit criteria for each work product
• Tracking defects to closure
• Recording process and quality data to improve both the review process and the software development process.

Inspection

An inspection is the most systematic and rigorous type of peer review. It follows a well-defined, multistage process with specific roles assigned to each participant. The most common inspection process includes seven stages: planning, overview, preparation, meeting, rework, follow-up and causal analysis.

participants other than the work product author lead the meeting (moderator), present the material to the inspection team (reader) and record issues as they're brought up (recorder).

Inspections are more effective at finding defects than informal review techniques. One telecommunications firm, I'm told, detected an average of 16 to 20 defects per thousand lines of code by inspection, compared to only three defects per thousand lines of code using informal reviews. The close scrutiny

of an inspection provides a thorough test of a product's understandability and maintainability and reveals subtle programming problems.

Diferents inspectores detectan diferentes problemas, pero la cantidad de problemas descubiertos no increasea llinearmente al sumar mas inspectores al equip de inspeccion

Team Review - Structured walkthrough

Son planned and structured, pero less formal and comprehensive

No hay reader role, no se ve de a pedazos el codigo, sino que se le pregunta a los participantes que comentarios tienen para hacer para una sección especifica o una pagina

Sin mas caros que los desk checks pero mas baratos que las inspections. Hay un recorder/scribe. Descubren aproximadamente 2/3 de los errores que descubre una inspección

Walkthrough

informal review donde el autor de un producto se lo describe a un grupo de peers y pide comentarios. El author es el que domina la meeting, a diferencia de las inspecciones. Aca se busca cumplir con las necesidades del autor, no los objetivos del grupo

No siguen u procedure ni require management reporting o metrics.

Descubren aproximadamente la mitad de lo que descubre una inspección

If you don't fully understand some information presented in a walkthrough, you might assume the author is right. It's easy to be swayed by an author's rationalization of his approach, even if you aren't convinced.

A reviewer's confusion is a clue that a defect lurks nearby

Pair Programming

Two developers work on the same producir simultaneously at a single workstation.

Esto facilita la comunicación y la oportunidad de informal reviews continuas y incrementales sobre las ideas y avances del otro.

“dos cabezas son mejor que una”

Promueve colaboración, collective ownership of the code y share commitment.

informal porque no tiene estructura ni proceso o documentación. No tiene perspectiva de alguien ajeno al código.

Peer Deskcheck

Solo una persona aparte del autor examina el trabajo-prodcto.

Depende del reviewer’s knowledge, skill and self-discipliine por lo que los resultados pueden ser muy variados

Cheapest review approach, ya que requiere el tiempo de un solo reviewer t quizás una follow up discussion con el autor del código.

Sirve para productos de bajo riesgo, y colleagues’ que tienen la habilidad de encontrar defectos rápido

Passaround

Como un peer deskcheck pero se le da una copia del código a muchas personas y se recibe el feedback.

Mejora el peer deskcheck porque cada persona tiene un skillset y un point of view distinto por lo que hay más probabilidad de que se encuentren defectos. Se pierde la estimulación de una group discussion.

Poner el código de forma electronica y dejar que los toros hagan comentarios evita redundancy y suma un poco de lo que se ganaría en una group discussion ya que los otros reviewers ven los comentarios que hicieron todos.

Ad hoc review

cosa del momento, le pedis a un compañero que le pegue una mirada y te diga lo que piense. Forma rápida de ganar otra perspectiva

Factores que aumentar el riesgo del producto

- Use of new technologies, techniques or tools
- Complex logic or algorithms that are difficult to understand but must be accurate and optimized
- Excessive developer schedule pressure
- Inadequate developer training or experience
- Mission- or safety-critical portions of the product with severe failure modes

- Key architectural components that provide a base for subsequent product evolution
- A large number of exception conditions or failure modes that must be handled, particularly if they're difficult to stimulate during testing
- Components that are intended to be reused
- Components that will serve as models or templates for other components
- Components with multiple interfaces that affect various parts of the product.

El riesgo del producto ayuda a determinar cual review approach es la mejor.

You might find that inspections work well for code, but team reviews or walkthroughs work best for design documents. Data provides the most convincing argument, but even subjective records of the kind of reviews you did, the work products that you examined and how well the reviews worked will be useful.

Review Activity					
Review Type	**Planning**	**Preparation**	**Meeting**	**Correction**	**Verification**
Inspection	Yes	Yes	Yes	Yes	Yes
Team Review	Yes	Yes	Yes	Yes	No
Walkthrough	Yes	No	Yes	Yes	No
Pair Programming	Yes	No	Continuous	Yes	Yes
Peer Deskcheck, Passaround	No	Yes	Possibly	Yes	No
Ad Hoc Review	No	No	Yes	Yes	No

Actividades que se realizan en los métodos

Review Objectives	Inspection	Team Review	Walkthrough	Pair Programming	Peer Deskcheck	Passaround
Find implementation defects	X	X	X	X	X	X
Check conformance to specifications	X	X			X	
Check conformance to standards	X				X	
Verify product completeness and correctness	X		X			
Assess understandability and maintainability	X	X		X	X	X
Demonstrate quality of critical or high-risk components	X					
Collect data for process improvement	X	X				
Measure document quality	X					

Metodo sugerido Segun el objetivo que busques

Exploratory Testing

test exploratorio → vas al producto a buscar errores.

> Exploratory Testing (ET): simultaneous learning, test design and test execution.

⚠ Any testing where the tester actively controls the design of the tests as they are performed, and uses the information gained while testing to design new and better tests
En cada momento debo pensar
cual es el mejor test que podría realizar, retroalimentándome de cad test que voy realizando

Also known as ad hoc testing. **unscripted testing.**

What makes exploratory testing interesting, and in my view profoundly important, is that when a tester has the skills to ***listen, read, think and report***, rigorously and effectively, without the use of pre-scripted instructions, the exploratory approach to testing can be many times as productive (in terms of revealing vital information) as the scripted variety.

Exploratory testing is not against the idea of scripting. In some contexts, you will achieve your testing mission better through a more scripted approach; in

other contexts, your mission will benefit more from **the ability to create and improve tests as you execute them.**

the process flows, and how it remains continuously, each moment, under the control of the practitioner.

La mayoría de las situaciones se benefician de mezclar scripted y exploratory testing

Basic external elements of ET: time, tester, product, mission, and reporting

Charter → da el contexto del test al tester, el environment, platforms y versions y la mission

Cosas que un exploratory tester tiene que poder hacer:

- Test Design → tiene que saber diseñar tests sistemáticamente
- Carful Observation → observar muy detalladamente para poder diseñar le test y poder mejorarlos
- Critical Thinking → poder review and explain their logic. Es importante para poder report status al final del testeo
- Diverse Ideas → creatividad
- Rich resources → muchas tools, information resources, test data, etc

Test Manager → hiring and firing authority.

Test Lead → focused on test strategy and tactics.

El charter me sirve para ver qué cosas tengo que probar. Aca la creatividad es cuándo la ejecuto a la prueba.

Oracle Testing

Como describir bugs de una forma acertada.

HICCUP

- History → un producto debería ser consistente con sus versiones anteriores. Si algo cambio, y nadie te dijo que debería haber cambiado, encontraste un problema
- Image → la imagen de la empresa. Si hay un bug que hace que la empresa se vea poco profesional, puedo argumentarlo así
- Comparable Product → compararlo con otro producto

- Claims → Si algo es inconsistente con lo que se “claimea” del producto
- User Expectation → el producto no hace algo que un usuario razonable se esperaría que haga, o no se hace de la forma que el usuario esperaría.
- Product → inconsistencia con el producto. Algo se comporta de una forma en una parte, pero de otra en otra parte
- Purpose → inconsistence con el propósito del producto. El behavior que encontré es contradictorio con lo que el usuario se espera.

Every time you log a new defect, before you click Send see whether you can answer the following questions:

Is this defect inconsistent with the product history?
Is this defect inconsistent with the image our company (or project team) is attempting to portray?
Is this defect inconsistent with a comparable product?
Is this defect inconsistent with claims made about the product?
Is this defect inconsistent with user expectations about the product?
Does this defect show an inconsistency within the product?
Is this defect inconsistent with the purpose of the product

Metricas

Metrica → Medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado

el indicador → métrica o combinación de métricas que ayuda con la toma de decisiones

KPI

objetivo → medir tal cosa.

es importante el método de medición, las fuentes de datos, la formula suada y como lo represento gráficamente

La frecuencia de cálculo debe ser tenida en cuenta y los valores esperados/normales definido.

Como implemento un programa de métricas

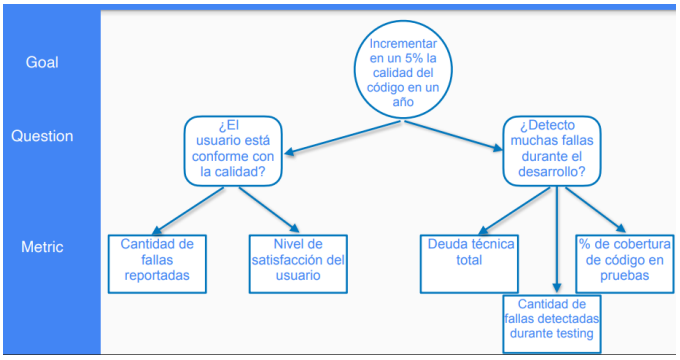
GQM → los primeros 3 pasos, los otros son de todos los programas de métricas. goals, questions, metrics

- Identificar los objetivos (**G**QM)
- Hallar las preguntas que me ayudan a alcanzar el objetivo (G**Q**M)
- Definir las métricas (GQM)
- Recolectar datos históricos
- Definir el proceso de recolección de datos para las métricas
- Recolectar validar y analizar los datos
- Usar métricas en la toma de decisiones

Goals → Aquello que la organización intenta alcanzar

Question → Aquellas preguntas cuyas respuestas permiten definir el cumplimiento de las metas

Metrics → las mediciones necesarias para ayudar a responder las preguntas y confirmar si las mejoras del proceso cumplieron su objetivo



Métricas orientadas a procesos

- Duración promedio de un proyecto
- Cantidad de proyectos segregados por tipo
- Esfuerzo promedio segregado por tipo/duración
- Defectos introducidos en una fase del ciclo de vida
- Defectos detectados en una fase del ciclo de vida
- % de tiempo/esfuerzo/costo dedicado a una fase del ciclo de vida
- % promedio de desvío en proyectos

    - Ev, PV, Schedule Variance, Cost variance, SPI, CPI

Métricas orientadas a Producto

- Cantidad de líneas de código (LOC)
- Funcionalidad (Puntos por Función / Use Case Points)
- Complejidad Ciclomática (Mc Cabe)
- Cohesión
- Acoplamiento
- Calidad de Producto (ISO 25000)
- Cantidad de fallas de un producto
- Confiabilidad (MTBF, MTTR)
  MTBF: Mean Time Between Failures
  MTTR: Mean Time To Recover

Métricas orientadas a recursos

- Cantidad de fallas detectadas por tester
- Cantidad de LOC (lines of code) producidas por un desarrollador
- Esfuerzo dedicado a codificar por un desarrollador
- Edad promedio del equipo
- Costo promedio de cada rol
- Años de experiencia promedio de cada rol

Conclusiones

> No podes controlar lo que no podes medir

Métricas representan datos objetivos → estamos o no atrasados y punto

NO MEDIR PERSONAS → caza de brujas. Si no amenazo puedo indagar y obtener información de porque la gente no llego a los objetivos

Detectar ls métricas más relevantes y medir pocos indicadores simples. SELECCIONARLAS

Sostener el esfuerzo de recolección en el tiempo

Usar las métricas para después tomar decisiones