


# Software Configuration Management

## Gestionando los cambios al Software



```
import { Router } from "vue-router";
import store from "../store/index";
import vuexI18n from "vuex-i18n";
import enLangFile from "../lang/en";

// Set config file into the global variable
window.config = require("../vue.config");

// Import bootstrap file
require("../bootstrap");

// Set Vue globally
const Vue = require("vue");

// Import Vue Router
const Router = require("vue-router");

// Import Vuex
const Vuex = require("vuex");

// Import Vuex I18n
const VuexI18n = require("vuex-i18n");

// Import enLangFile
const enLangFile = require("../lang/en");

// Create Vue instance with router, store, and i18n
const app = new Vue({
  router: Router,
  store: store,
  i18n: new VuexI18n({
    locale: "en",
    messages: {
      en: enLangFile,
    },
  }),
}).$mount("#app");
```

# Temas

- ▶ Conceptos de Software Configuration Management (SCM)
- ▶ Administración del proceso de Gestión de Configuración del Software
- ▶ Identificación de la configuración
- ▶ Control de Cambios de la configuración
- ▶ Status Accounting de la Configuración
- ▶ Auditorías de la Configuración de Software
- ▶ Release Management
- ▶ Software Configuration Management Tools
- ▶ Devops

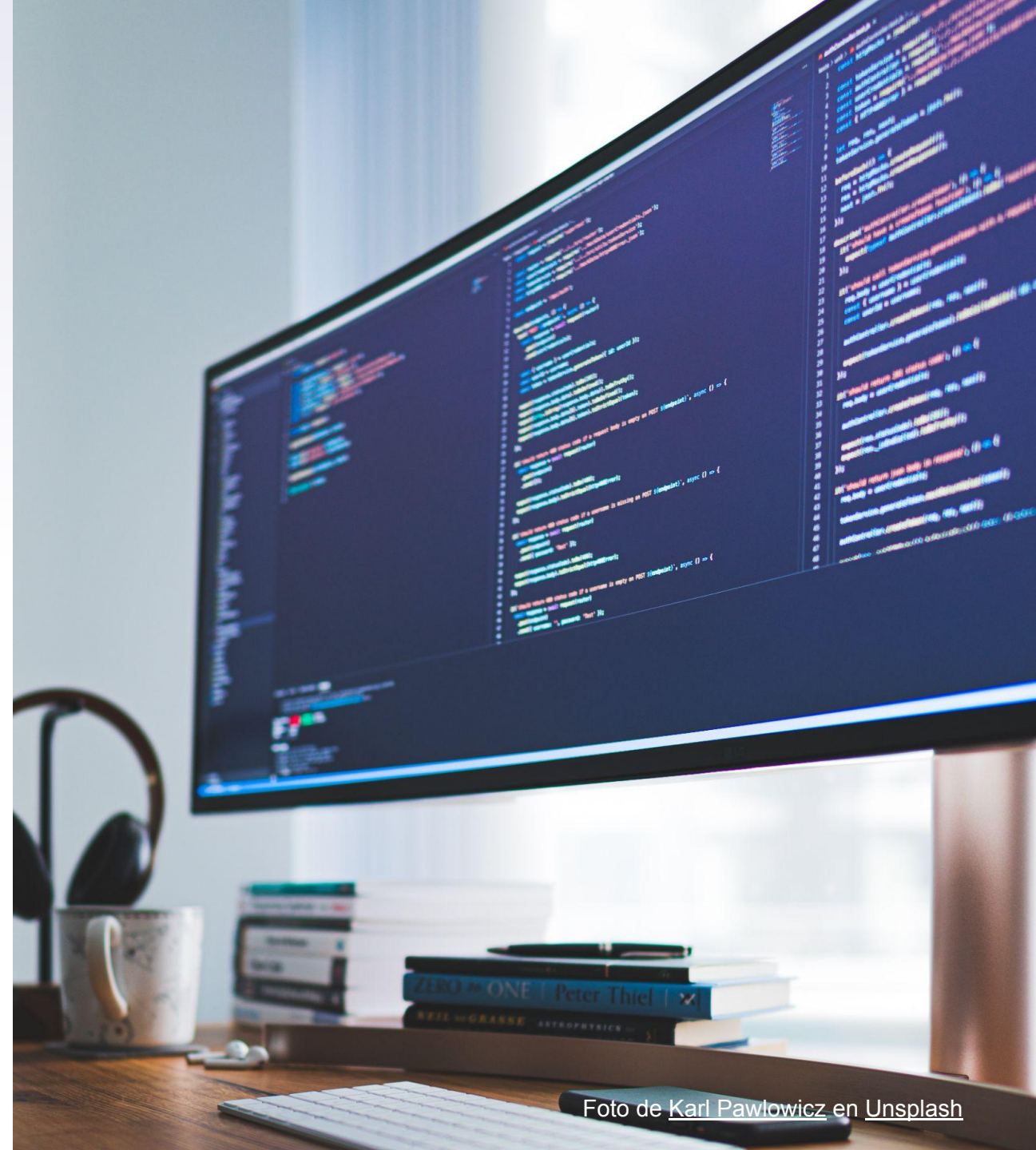
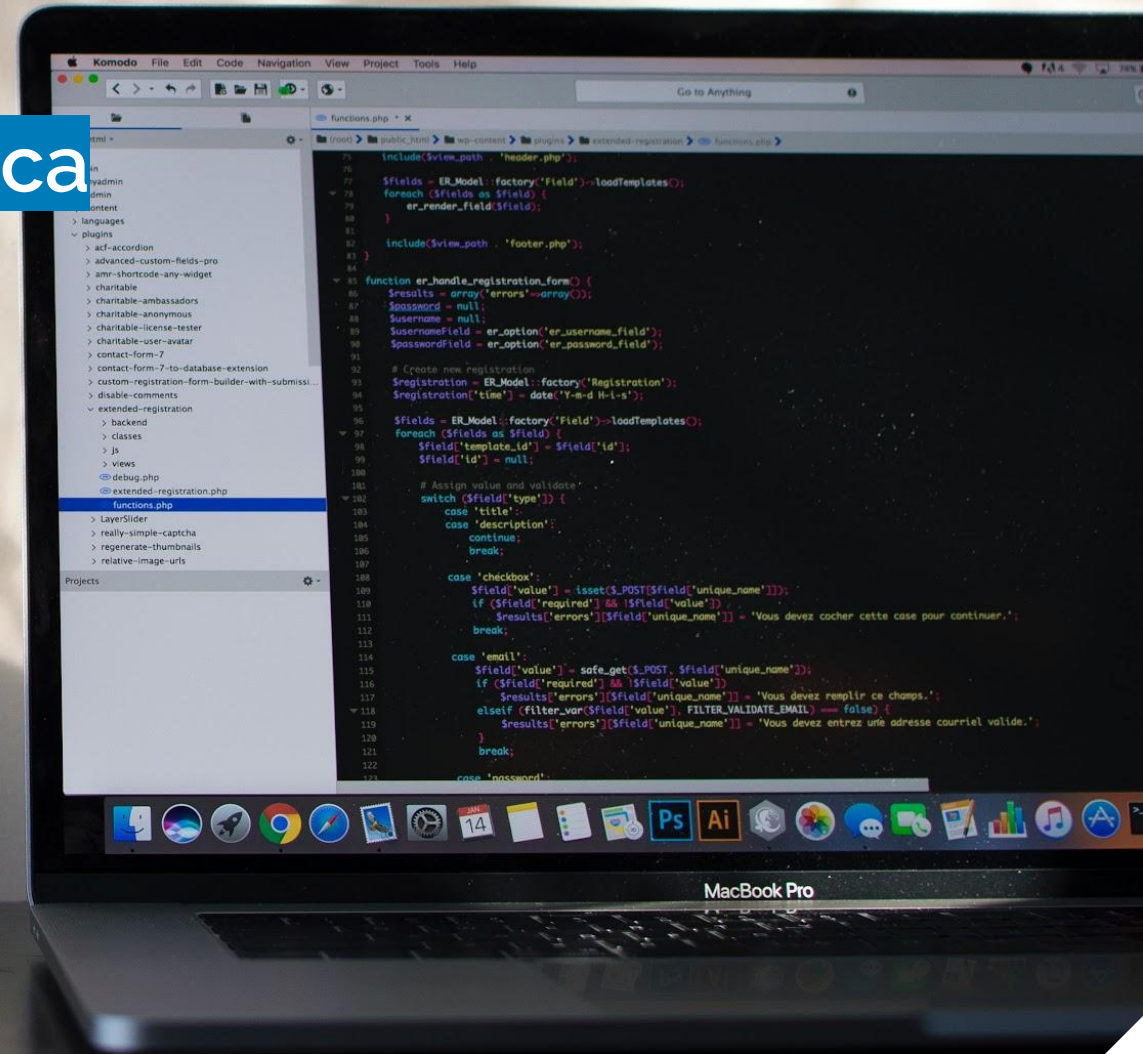


Foto de [Karl Pawlowicz](#) en [Unsplash](#)



# Conceptos de la Gestión de Configuración del Software

Introduciéndonos en la problemática



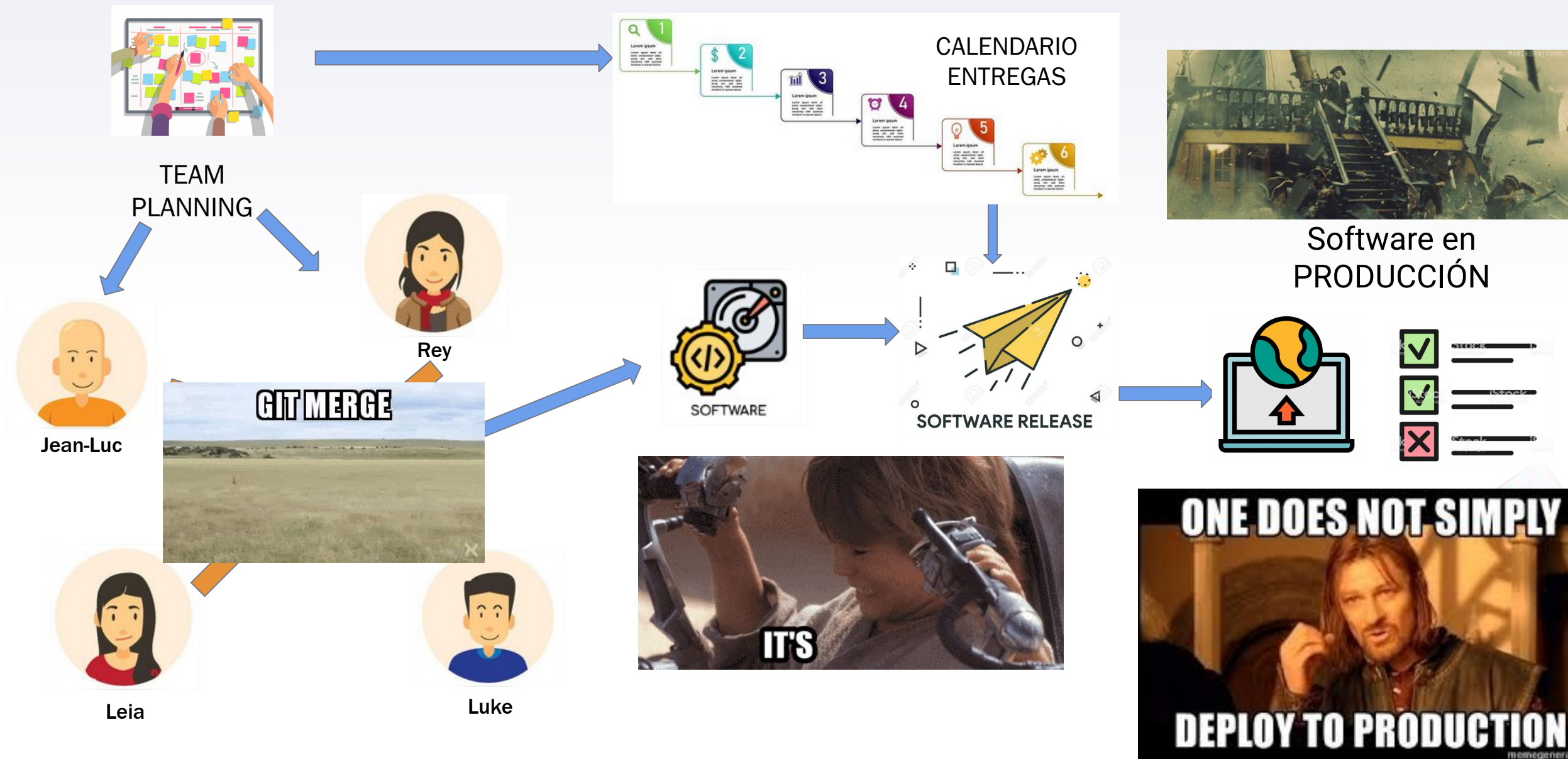
# Problemas en el desarrollo de Software


- ▶ "En mi maquina me funciona"
- ▶ "Actualizamos la API. No sabíamos que todavía había usuarios con v1.0"
- ▶ "Armar el entregable toma unos días"
- ▶ "En qué clientes estaba el bug?"
- ▶ "Pasa que probaste en el entorno de testing y no en desarrollo..."





# El problema de cualquier compañía con + de 1 dev

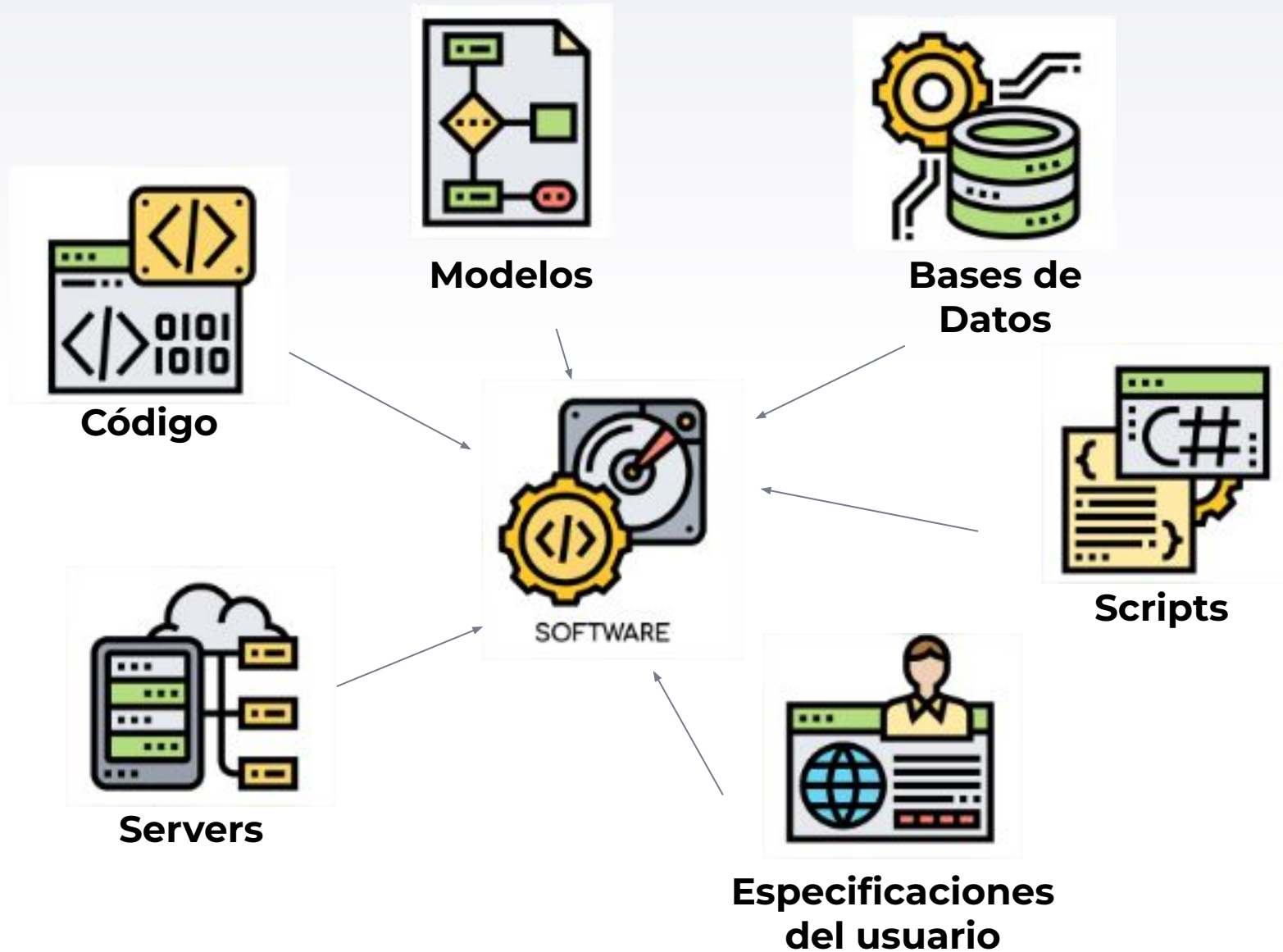




La **Gestión de Configuración de Software** es una disciplina orientada a administrar la evolución de **Productos, Procesos y Ambientes**, definiendo mecanismos para administrar diferentes versiones de los mismos, controlando los cambios efectuados, y auditando y reportando la actividad de cambios realizada

Su **propósito** es establecer y mantener la integridad de los artefactos del proyecto de software a lo largo del ciclo de vida del mismo.

# ¿Pero qué es el Software?





# Conceptos clave de la gestión de configuración de software



# Conceptos clave

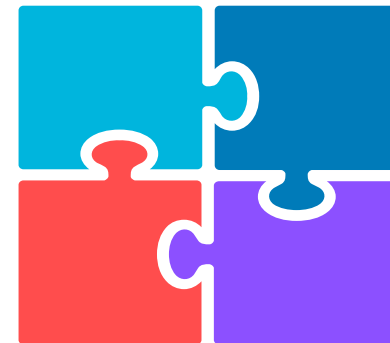
## Ítem de Configuración (IC)

- ▶ Es cualquier elemento involucrado en el desarrollo del producto y que está bajo el control de la gestión de configuración
- ▶ Para cada ítem se conoce información específica para poder identificarlo, como por ejemplo nombre, versión, fecha de creación, autor, entre otros atributos.



## Configuración (De Software)

- ▶ Es el conjunto de ítems de configuración que gestionaremos para la construcción de nuestro producto Software
- ▶ La Configuración de Software puede o no contener código, dependerá del momento en la construcción del producto Software.
- ▶ **Ejemplo:** Al iniciar un proyecto, la Configuración puede estar compuesta únicamente por una Especificación de Requerimientos Software



# Conceptos clave

## IC's de Proceso

- ▶ Son los ítems generados por el proceso de desarrollo. Ejemplos:
  - ▷ Plan de CM
  - ▷ Propuestas de Cambio
  - ▷ Plan de desarrollo
  - ▷ Plan de Calidad
  - ▷ Lista de Control de entrega
  - ▷ Planes de proyecto
- ▶ Algunos de estos ítems sólo son gestionados durante una parte de la vida del producto sw, ejemplo, el plan de proyecto solo es gestionado durante el proyecto y luego se deja de gestionar o se elimina
- ▶ Otros ítems de proceso pueden seguir siendo gestionado durante toda la vida del producto sw, ejemplo el plan de CM

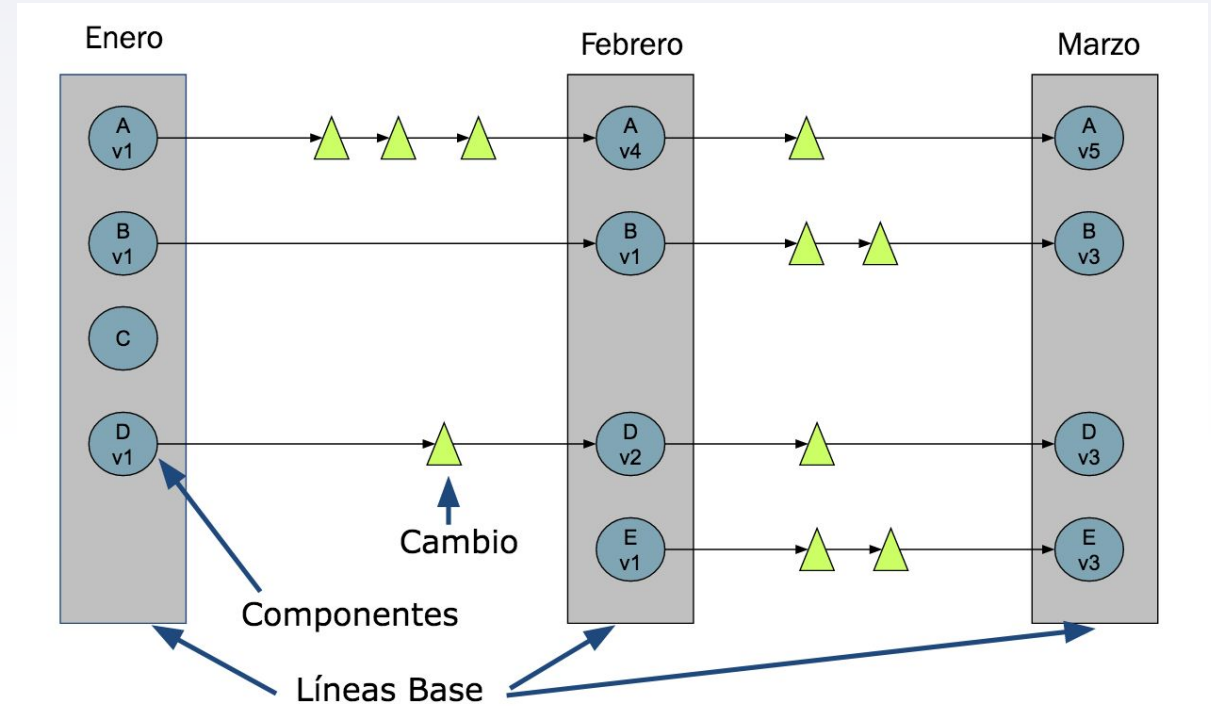
## IC's de Producto

- ▶ Son los ítems propios del producto software y que son gestionados durante todo el ciclo de vida del mismo. Son transversales a todos los ciclos de mantenimiento del producto. Ejemplos:
  - ▷ Requerimientos
  - ▷ Plan de Integración
  - ▷ Manual de Usuario
  - ▷ Arquitectura del Software
  - ▷ Estándares de codificación
  - ▷ Casos de prueba
  - ▷ Código fuente
  - ▷ Documento de despliegue

# Conceptos clave

## Línea Base (Baseline)

- ▶ Representa un estado de la 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 esta configuración del ítem o conjunto de ítems satisface (n) algunos requerimientos funcionales o técnicos.
- ▶ Esta línea base puede no contener código y puede no coincidir con un release de Software



*IEEE (IEEE Std. No. 610.12-1990) define a la línea base como "una descripción y revisión acordada de los atributos del producto, que luego sirven como base para un mayor desarrollo y definición del cambio, y este cambio solo se puede realizar a través de procedimientos formales de control de cambios".*



# Conceptos clave

## Trazabilidad

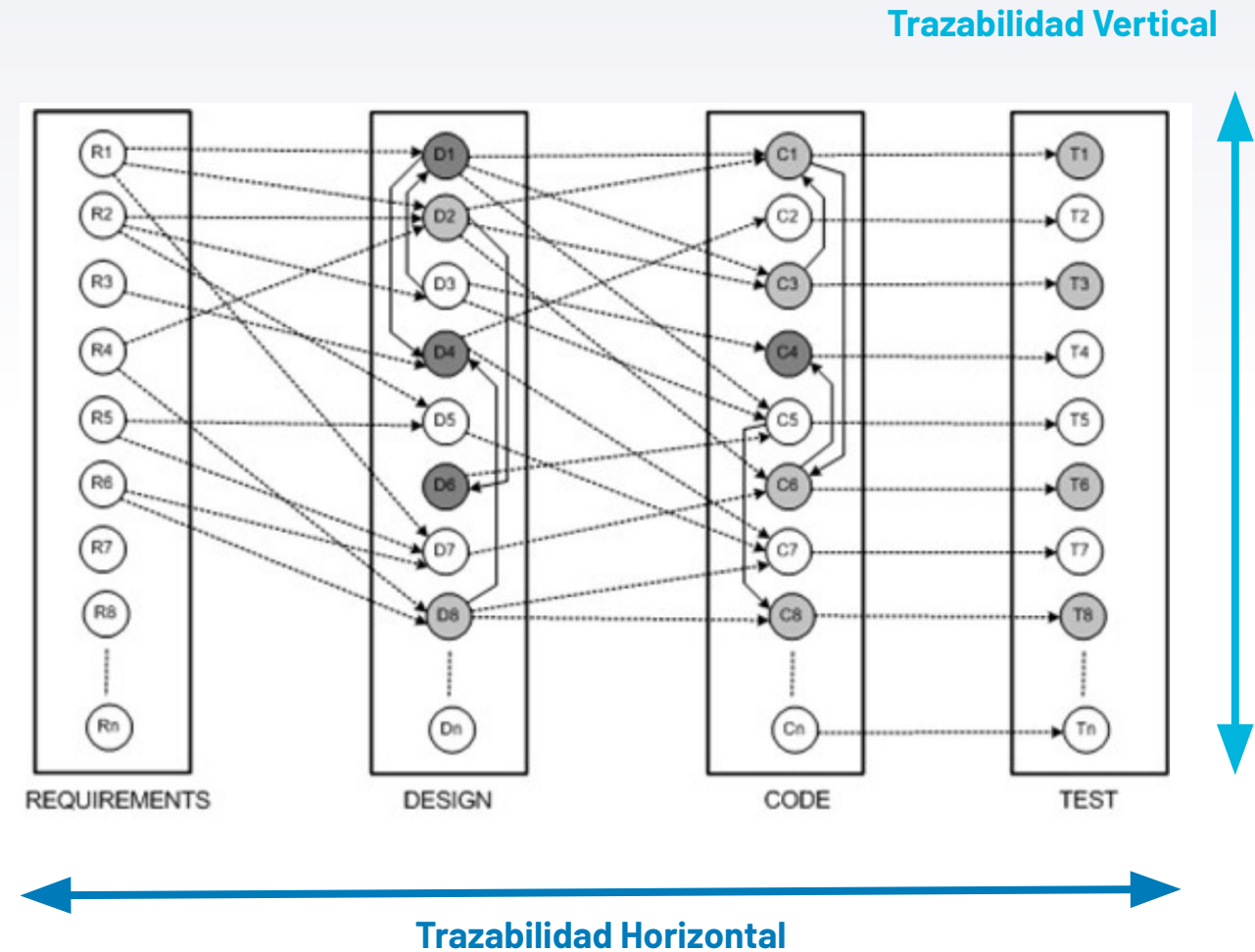
- ▶ Es la capacidad de rastrear un ítem de configuración a lo largo de todo el proceso

## Trazabilidad Horizontal

- ▶ Representa la capacidad de trazar todos los ítems de configuración generados en cada etapa del proceso por un mismo requerimiento.

## Trazabilidad Vertical

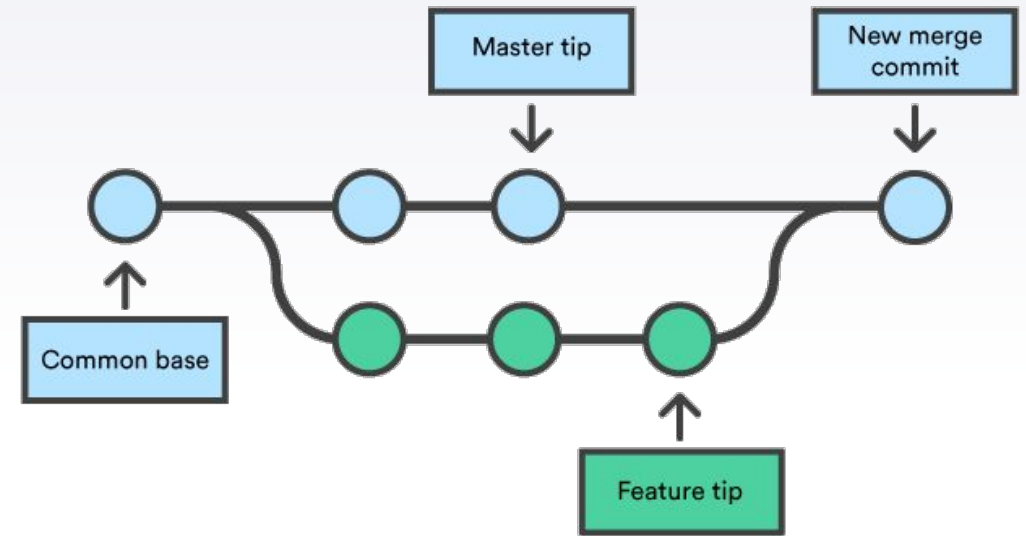
- ▶ Representa la capacidad de trazar todos los ítems de configuración del mismo tipo, generados en la misma etapa



# Conceptos clave

## Branching

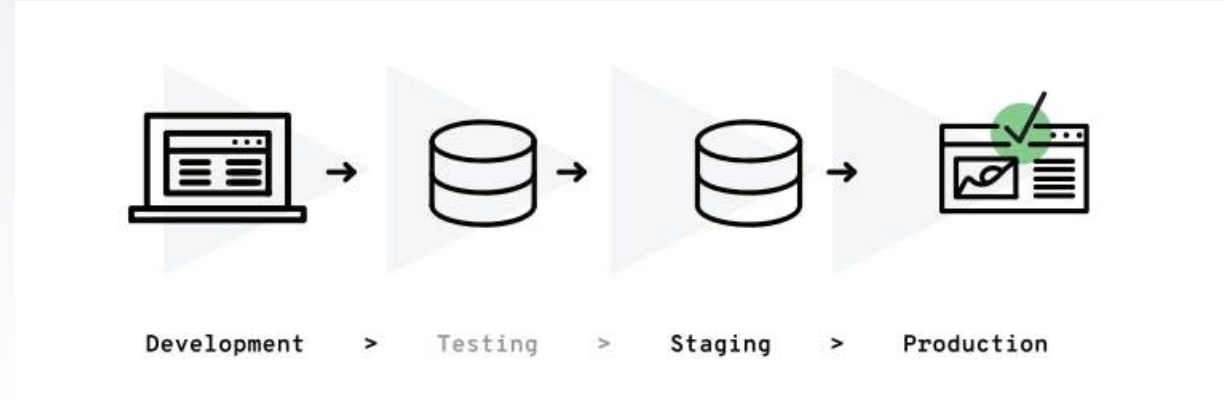
- ▶ Es la acción de crear líneas de desarrollo separadas de una línea de trabajo. Se las conoce usualmente como **branches**
- ▶ Estos branches son mecanismos utilizados por los equipos para que puedan desarrollar nuevas funcionalidades utilizando un entorno de trabajo separado del resto.
- ▶ Estas líneas utilizan las líneas base de un repositorio existente como punto de partida, de esta forma, las features y bug fixes son desarrolladas de forma separada que facilitando el trabajo de estos temas en paralelo.
- ▶ Permite a los miembros del equipo trabajar en múltiples versiones de un producto, utilizando el mismo set de Ítems de Configuración



# Conceptos Clave

## Ambiente (Environment)

- ▶ Un entorno de implementación es una colección de servidores, clústeres y servicios configurados que colaboran para proporcionar un entorno para alojar módulos de software.
- ▶ Cada entorno se construye con un propósito, pudiendo tener características funcionalmente similares pero no necesariamente ser iguales a nivel infraestructura.
  - ▶ Un ejemplo asociado a esto son los entornos de pre-producción y producción. No necesariamente son iguales a nivel infraestructura pero deberían ser equivalentes a nivel software.
  - ▶ El propósito del ambiente es decidido por la organización o el equipo de trabajo. Es decir cada organización o equipo decide para qué será utilizado el ambiente y en qué etapa del proceso de desarrollo se utilizará.
- ▶ Ambientes comunes en el desarrollo suelen ser desarrollo, staging, producción







## Funciones de la gestión de configuración de Software (Acorde al SWEBOK)

# Funciones de SCM (Acorde a SWEBOK)

## Administración del Proceso SCM

- Contexto Organizacional de SCM
- Plan de SCM
- Vigilancia del Proceso SCM



## Identificación de la CS

- Identificar Ítems a ser Controlados
- Identificar ítems de 3ros o bibliotecas



## Control de la CS

- Solicitar, evaluar y aprobar cambios al SW
- Implementar cambios al SW
- Desviaciones y excepciones del proceso



## SC Status Accounting

- Información de Estado de la Gestión de CM
- Reportes de Estado de la Configuración



## Auditoría de la Configuración

- Auditoría Funcional
- Auditoría Física
- Auditoría de Proceso

## Release Management & Delivery

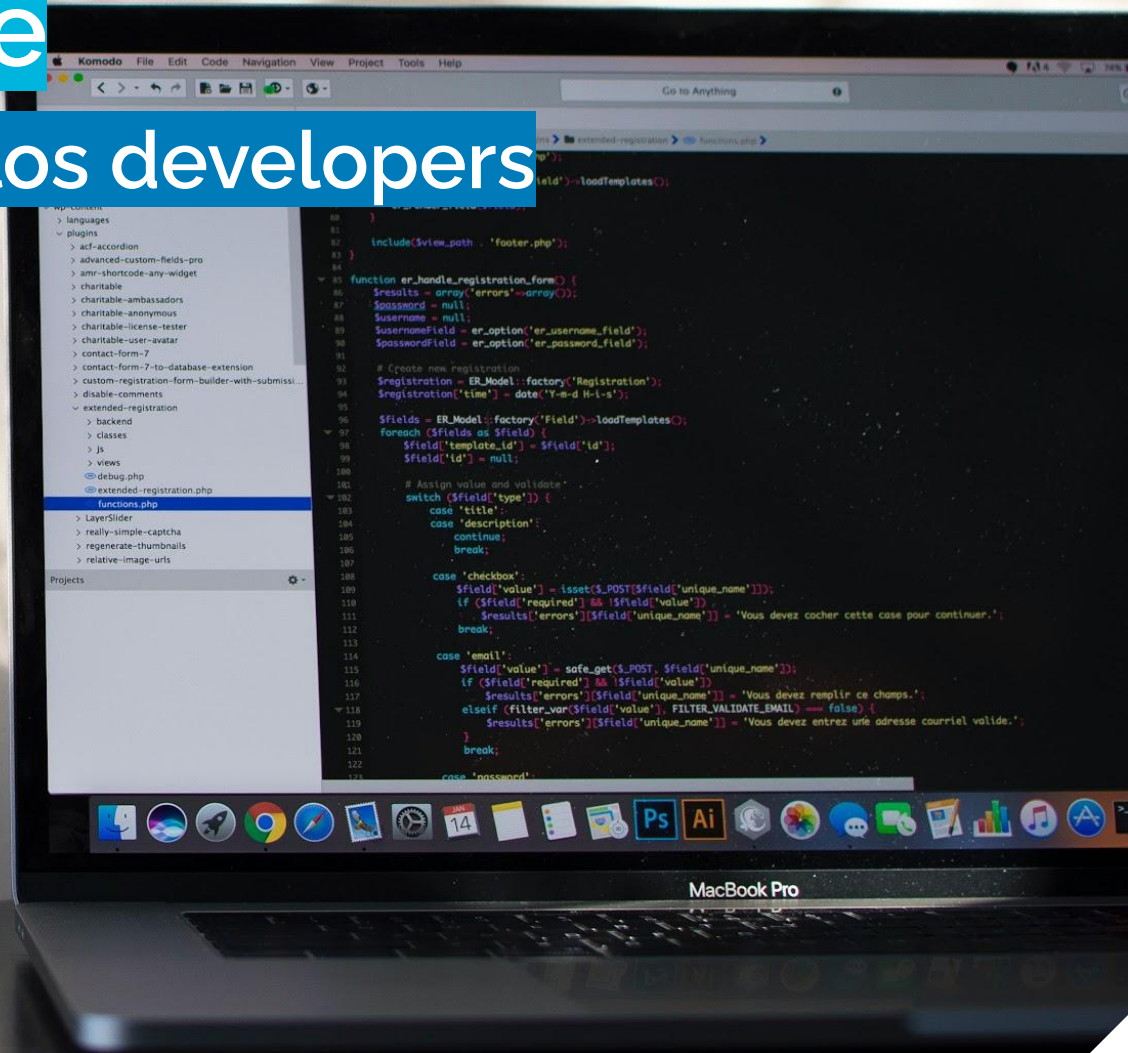
- Software Building
- Software Releasing

## SCM Management Tools


- Herramientas para soportar los diferentes aspectos y niveles de la gestión de configuración

# Administración del proceso de Gestión de Configuración del Software

## Definiendo el marco de trabajo de los developers







La **Administración del Proceso de Gestión de Configuración del Software** define herramientas, reportes y procesos que serán utilizados, en la organización o el proyecto, para la gestión de configuración.

Se encarga de definir cómo serán incorporados los ICs, cuando se realizarán las líneas base, cómo será la estrategia de desarrollo en paralelo, y cómo serán nombradas las versiones, entre otras definiciones.

Generalmente se realiza mediante la definición de un plan de CM o Plan de Gestión de Configuración



# Planning for Software Configuration Management

# Plan de gestión de configuración (CM Plan)

Un plan de SCM define principalmente:



Lista ítems de configuración (IC), y en qué momento ingresan al sistema de CM



Define estándares de nombres, jerarquías de directorios, estándares de versionamiento



Define las estrategias de Branching



Define los procedimientos para crear "builds" y "releases"



Define 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)



# Proceso de SCM definido

Podemos decir que tenemos un proceso definido cuando (Por ejemplo):

Está definido cómo se registran cambios de software



Se establecieron los roles que aprueban o rechazan el cambio



**Jean-Luc**  
analiza y decide si se hace o no junto con el PO

Están identificados los repositorios de los artefactos



como repositorio de código



como gestor de paquetes

Hay establecida una política de trabajo sobre esos repositorios



Luke



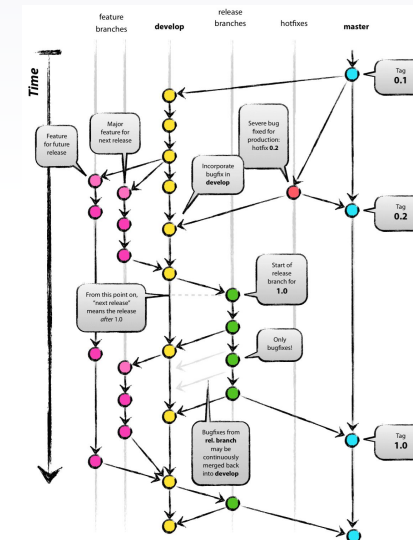
Rey



Leia

El equipo conoce las reglas de desarrollo

Están identificados los momentos en los que se realizan las líneas base



gitflow como flujo de trabajo

Los cambios son gestionados completamente, desde el ingreso hasta su puesta en producción





**Key Concept: Branching Strategies**

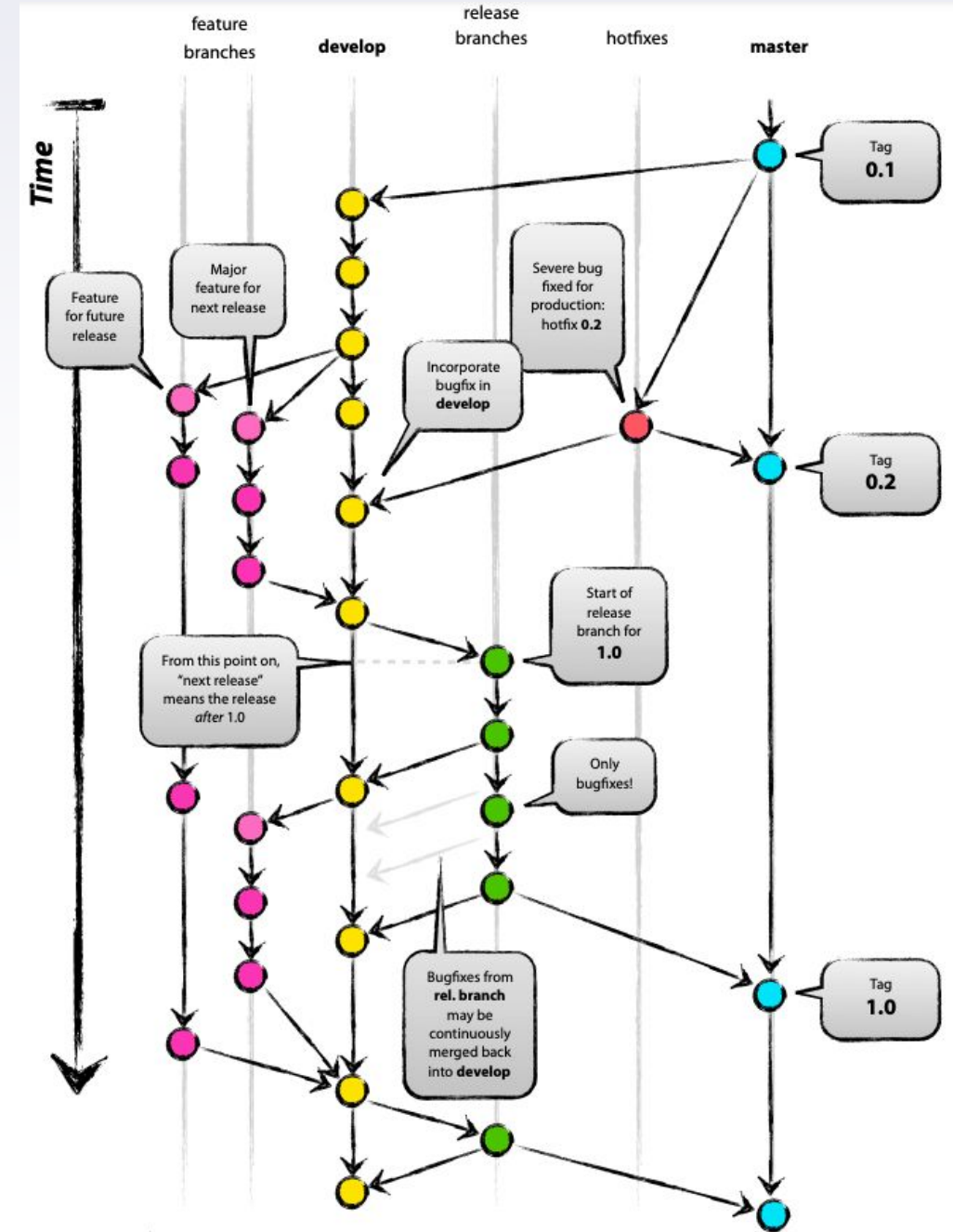
# Branching Strategies

- ▶ La estrategia de Branching es la estrategia que adoptan los equipos de desarrollo de software al escribir, mergear e implementar código cuando usan un sistema de control de versiones (git / SVN / TFS).
- ▶ Es esencialmente un conjunto de reglas que los desarrolladores pueden seguir para estipular cómo interactúan con una base de código compartida.
- ▶ Una estrategia de branching tiene como objetivos:
  - ▷ Mejorar la productividad al garantizar una coordinación adecuada entre los desarrolladores
  - ▷ Facilitar el desarrollo paralelo
  - ▷ Ayudar a organizar una serie de lanzamientos planificados y estructurados
  - ▷ Trazar un camino claro al realizar cambios en el software hasta la producción
  - ▷ Mantenga un código libre de errores donde los desarrolladores puedan corregir problemas rápidamente y hacer que estos cambios vuelvan a la producción sin interrumpir el flujo de trabajo de desarrollo.

# gitflow

## ► Concepto

- Consiste en dos ramas principales que existen a lo largo de todo el ciclo de vida del desarrollo, master y develop y tiene una serie de branches de soporte. En Master vive el código productivo.

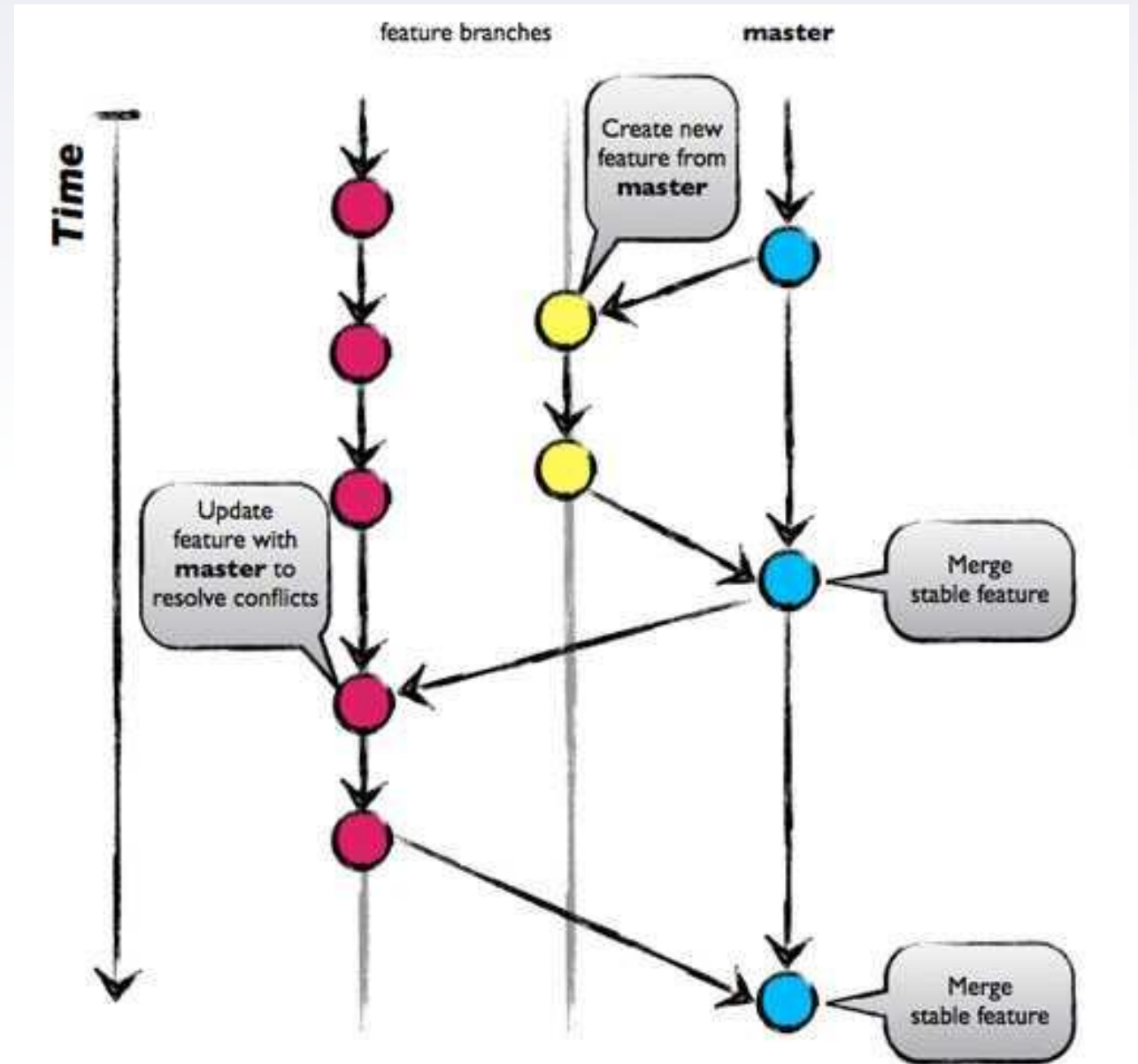




# github flow

## ► Concepto

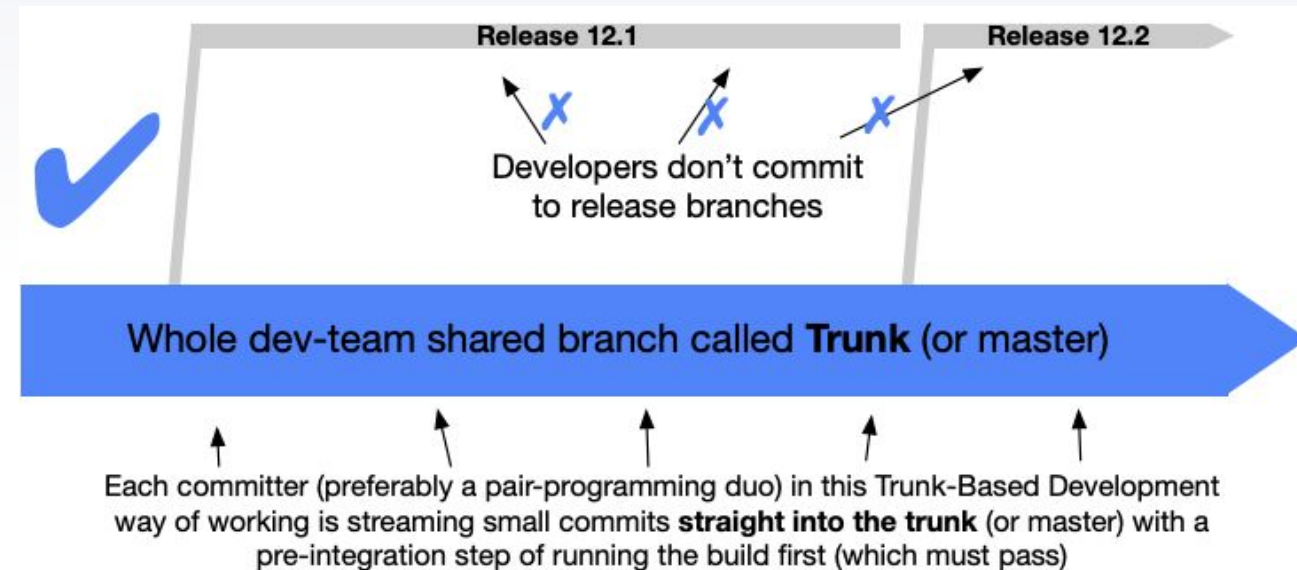
- Creado por GitHub, busca ser una alternativa simplificada del desarrollo,
- Existe una rama Master de donde parten las ramas de desarrollo
- Cada nueva feature se trabaja en una rama separada que es integrada nuevamente en master cuando el trabajo finaliza



# trunk based flow

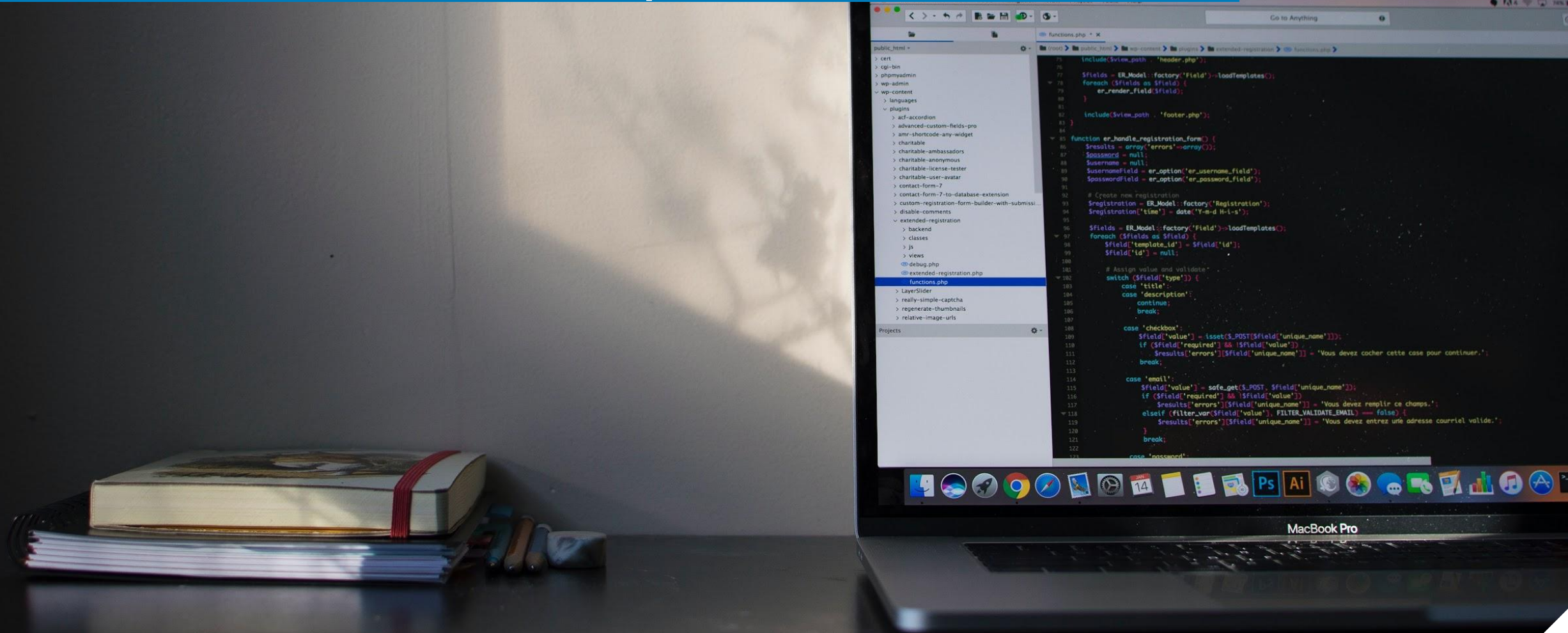
## ► Concepto


- Un modelo de branching de control de fuente, donde los desarrolladores colaboran en el código en una sola rama llamada "troncal" \*, resiste cualquier presión para crear otras ramas de desarrollo de larga duración mediante el empleo de técnicas documentadas. Por lo tanto, evitan fusionar el infierno, no rompen la construcción y viven felices para siempre.



# Identificación de la configuración

## Establecer como esta compuesto nuestro Software





La **identificación de la configuración** es la actividad de definir qué elementos (ICs) estarán controlados por la gestión de configuración ya que no todos los artefactos necesitan ser gestionados para garantizar la integridad del producto.

Para esto, la actividad establece guías o criterios para entender qué elementos son parte o no de una configuración.

La **identificación de la configuración** también define momentos o condiciones para establecer una línea de base o bien para liberarla (es decir llevarla a otro ambiente, también conocido como “release”)



# Identificación de la configuración

La actividad tiene que identificar los ítems de configuración que serán gestionados. Para ello, se pueden ejecutar distintas acciones, como por ejemplo:

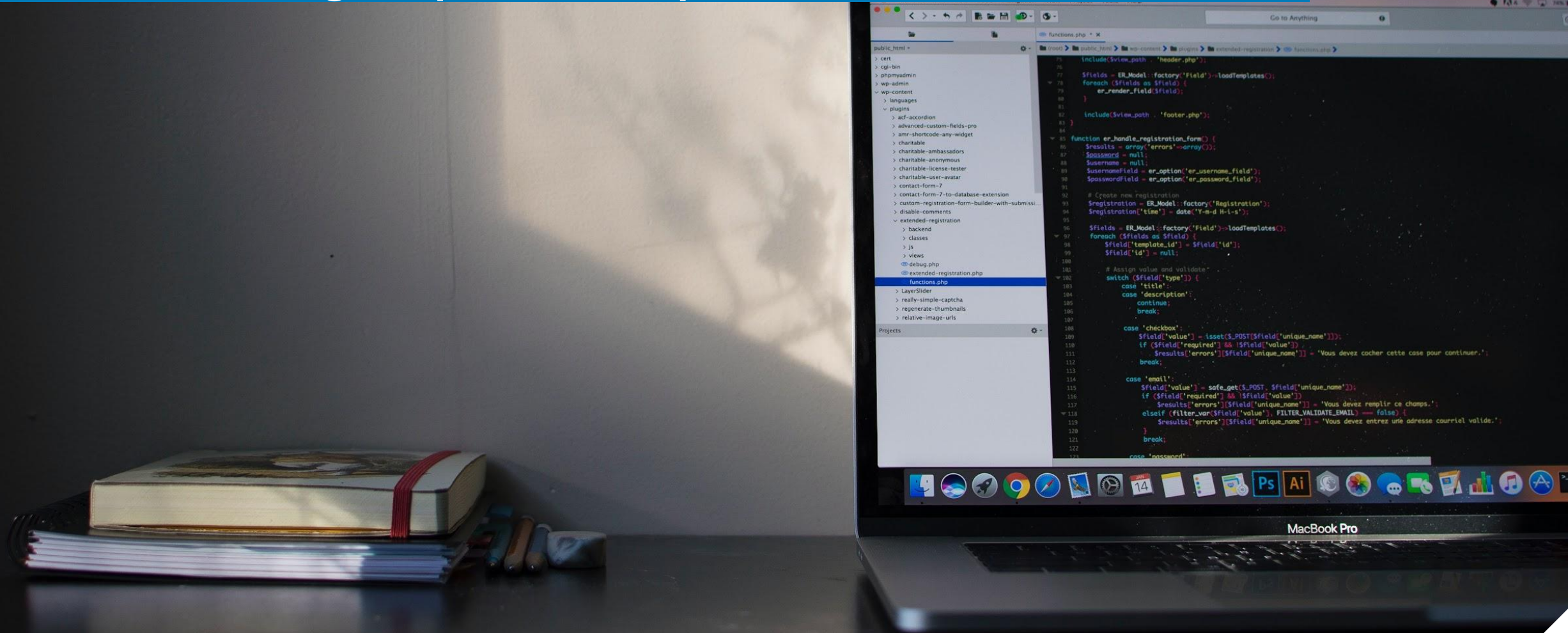
- ▶ **Identificar Código**
  - ▶ Buscar fuentes
  - ▶ Chequear repositorios
  - ▶ Entender la arquitectura
- ▶ **Identificar ambientes de ejecución**
  - ▶ Tratar de instalar una versión
  - ▶ Identificar Bases de datos, servers, configuraciones
- ▶ **Identificar Documentación**
  - ▶ Buscar especificaciones
  - ▶ Entender cambios anteriores

# ¿Qué elementos se registran de un IC?

- ▶ En general, es importante identificar para cada IC los siguientes atributos:
  - ▷ Nombre
  - ▷ Versión
  - ▷ Autor / Revisores
  - ▷ Módulos o ítems relacionados
  - ▷ Última modificación
  - ▷ Tipo de IC (código, scripts, diagramas, requerimientos, etc.)
- ▶ Estos atributos serán registrados y se utilizarán como información útil en la actividad de Status & Accounting de la configuración

# Control de Cambios de la configuración

## Establecer reglas para incorporar cambios al Software





El **Control de Cambios de la Configuración** asegura que los ítems de configuración mantienen su integridad ante los cambios a través de:

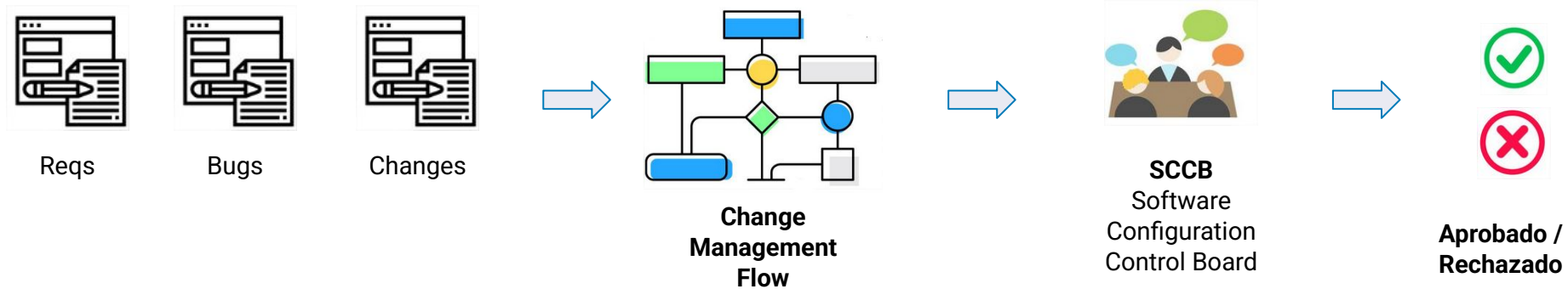
- La **identificación** del propósito del cambio
- La **evaluación del impacto** y aprobación del cambio
- La **planificación** de la incorporación del cambio
- El **control** de la implementación del cambio y su verificación

El Control de Cambios de la Configuración consiste en establecer un procedimiento de control de cambios y controlar el cambio y la liberación de ICs a lo largo del ciclo de vida.



# Software Configuration Control Board

- ▶ Es quién tiene la autoridad de aceptar o rechazar un cambio en la configuración de Software
- ▶ En proyectos pequeños puede residir en el Líder del proyecto, y a medida que se incrementa la complejidad, puede ir incorporando nuevos integrantes
- ▶ Puede haber diversos niveles de autorización de acuerdo a diferentes criterios
  - Por ejemplo: Un nivel puede ser compuesto solo por Stakeholders para autorizar o no un cambio de alcance. mientras que en otro nivel el Product Owner acepta o no un cambio en el backlog del sprint



# Change Management Process

- ▶ Es el flujo y las herramientas definidas para identificar, analizar y aprobar cambios a una configuración. Difiere en cada Organización, pero en general se puede dividir en las siguientes etapas:
- ▶ **Identificar el cambio**
  - ▷ Incluye formalizar el pedido de cambio
- ▶ **Evaluar el impacto**
  - ▷ Analizar el cambio y determinar el impacto que tendrá en el software, el equipo que trabaja en él y quienes lo usarán. Evalúe los aspectos técnicos del cambio, sus posibles efectos secundarios y el impacto general en otras partes del sistema.
- ▶ **Decidir hacer o no el cambio**
  - ▷ Entendiendo el impacto en el negocio, o en el calendario o en el equipo.
- ▶ **Planificar la implementación del cambio**
  - ▷ Incorporarlo dentro de las tareas a realizar

# Ejemplo Proceso Control de Cambios

- ▶ **Se solicita el cambio**
  - ▶ Puede ser solicitado tanto por un usuario como un desarrollador. En general, solo se aceptan cambios de personas autorizadas a pedirlo.
- ▶ El **cambio es evaluado** y comparado contra los objetivos del proyecto
  - ▶ Un grupo de asesores evalúa el cambio y lo aprueba o rechaza
  - ▶ Se suele evaluar el impacto del cambio tanto en el producto como en la calendarización del proyecto
- ▶ Si se **rechaza**, se puede dejar como parte de una “Segunda Fase” del proyecto, o incorporarlo en una segunda iteración de desarrollo.
- ▶ Si es **aceptado**, se re planifican las tareas a realizar y se le asigna recursos para resolverlo
- ▶ El cambio implementado deberá ser revisado en auditorías.
  - ▶ La complejidad de la administración del cambio varía de acuerdo al proyecto. En proyectos pequeños, el pedido de cambio puede ser informal y desarrollarse rápidamente mientras que en proyectos más complejos puede haber comités de varias personas que evalúen el cambio

# Mediciones relacionadas a Change Requests

## Successful Changes

Cuenta la cantidad de cambios que han sido aprobados y desplegados en producción sin generar incidentes

## Emergency Changes

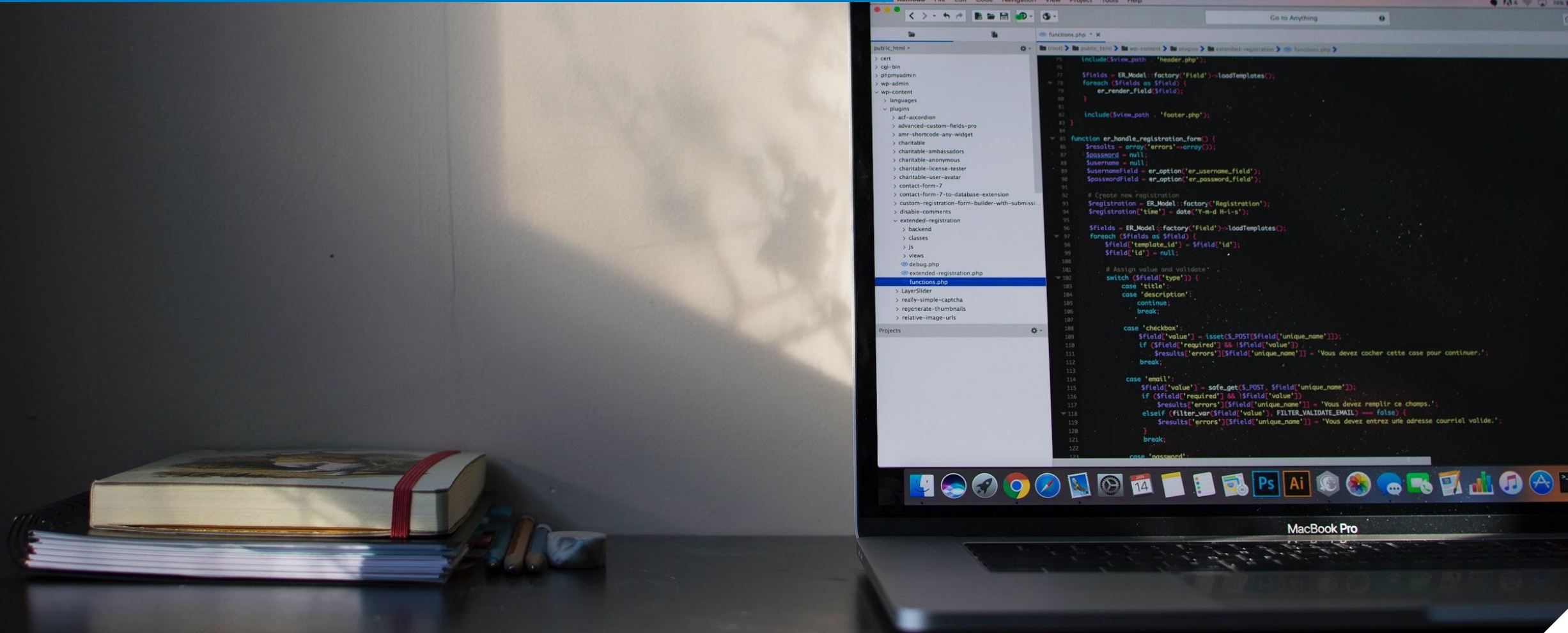
Cuenta la cantidad de cambios que han sido aprobados por canales de urgencia o emergencia.

## Changes in Backlog

Mide la cantidad de cambios pendientes de ser analizados. Es un indicio de la eficiencia del proceso de aprobación de cambios y se puede utilizar para evaluar la velocidad del equipo

# Status Accounting de la Configuración

## Analizando la historia de los cambios







**Status Accounting** tiene la función de registrar y reportar la información necesaria para administrar la configuración de manera efectiva. Para ello debe:

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

Una implementación exitosa de esta actividad debe poder contestar:

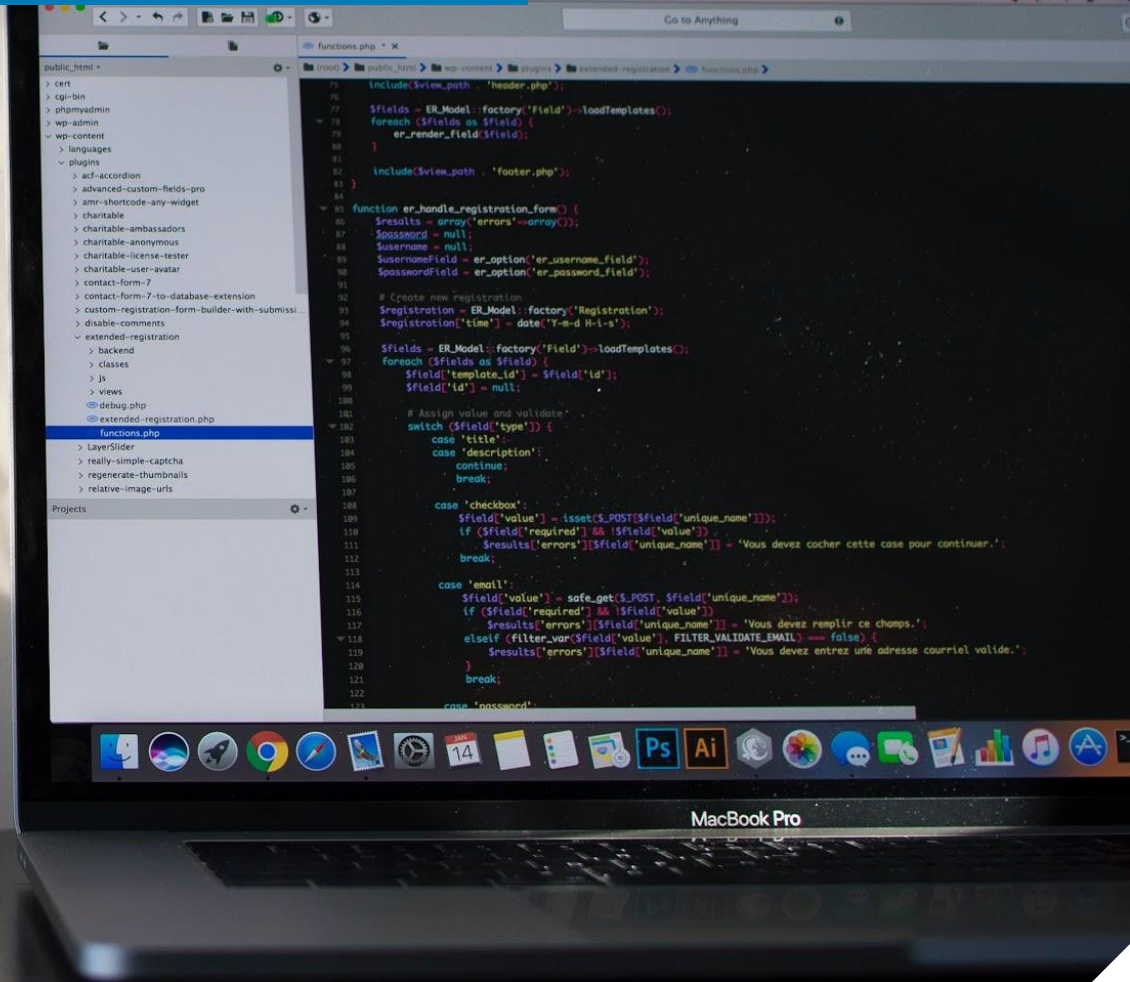
- ¿Qué cambios se realizaron al sistema?
- ¿Cuándo se realizaron dichos cambios?
- ¿Quién lo cambió?
- ¿Qué cambió?


También responder acerca del proceso de cambios

- ¿Quién aprobó el cambio?
- ¿Quién solicitó el cambio?

# Auditorías de la Configuración de Software

## Validando los distintos aspectos de la Gestión de Configuración de Software





De acuerdo al SWEBOOK, una **auditoría de Configuración de Software** es una examinación de un trabajo, producto o set de artefactos para evaluar aspectos técnicos, de seguridad, legales y de cumplimiento de normas con respecto a especificaciones, estándares, acuerdos contractuales u otros criterios.

Las auditorías son llevadas a cabo de acuerdo a procesos definidos y pueden ser ejecutadas con diferentes niveles de formalidad, desde Revisiones informales basadas en checklists hasta Pruebas exhaustivas de la configuración que son planificadas con anticipación.

# Tipos de Auditorías

## Auditoria Funcional

- ▶ Es una evaluación del producto software que se realiza para verificar, vía testing, inspección demostración o análisis de resultados, que el producto ha cumplido los requerimientos especificados en la línea base de documentación funcional.
- ▶ En las auditorías funcionales, básicamente se verifica que una configuración dada cumpla con alguna especificación de requerimientos previamente definida.

## Auditoria Física

- ▶ Evalúa que los elementos modificados de la configuración sean consistentes con las especificaciones técnicas de los cambios en la versión.
- ▶ Esta auditoría también evalúa los cambios realizados en el código y en los diferentes ambientes

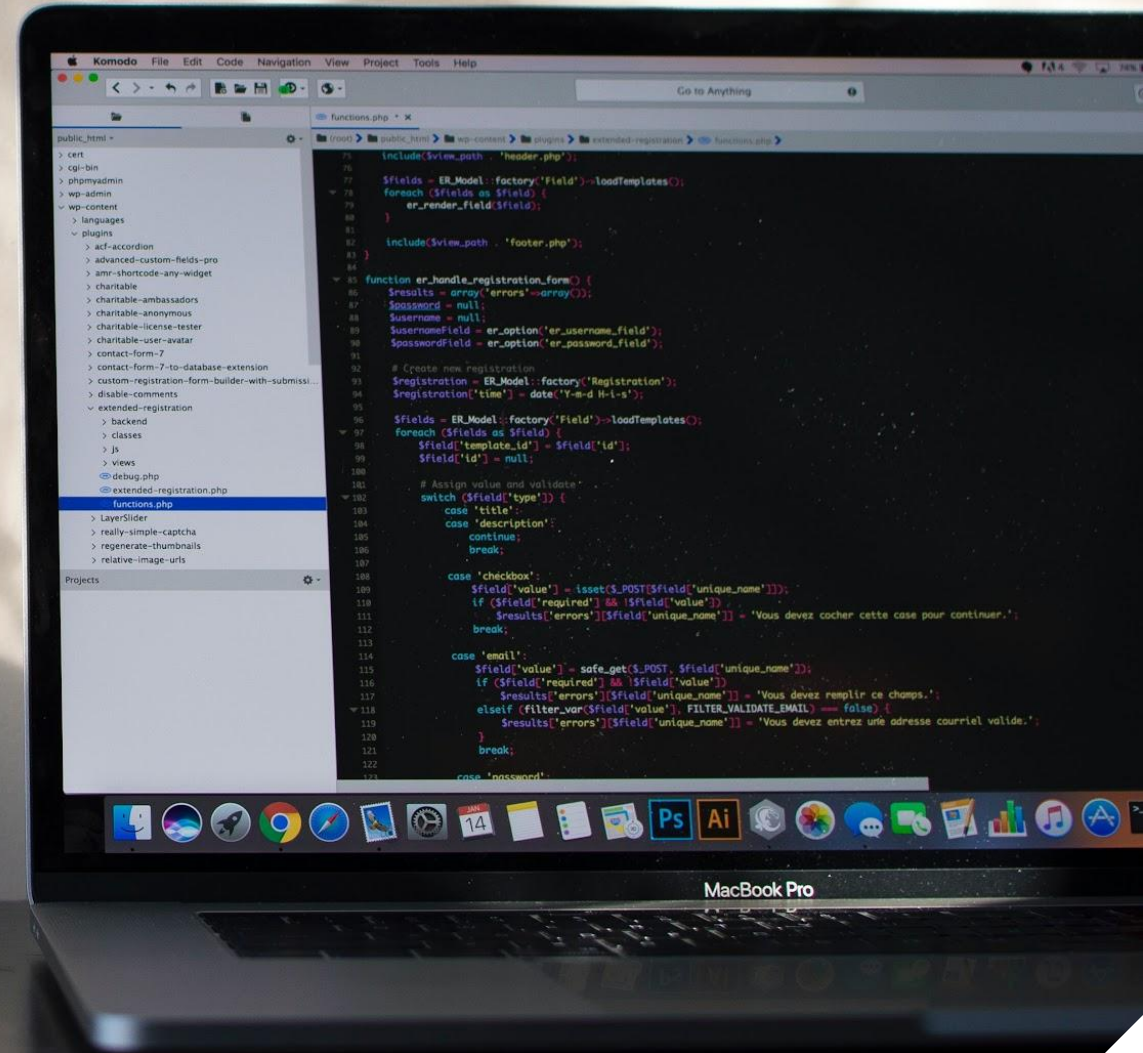
## Auditoria de Proceso

- ▶ Evalúa que los cambios realizados al producto software hayan seguido el proceso definido para los mismos
- ▶ El objetivo es confirmar que todos los cambios fueron solicitados, aprobados e implementados de acuerdo al proceso definido en la compañía
- ▶ Ejemplo:
- ▶ Al identificar una línea base se identifica su composición (qué ítems la conforman)
- ▶ Las inconsistencias se reportan y se determina si el defecto corresponde a la configuración misma o a la documentación




# Release Management

## Building & Releasing a Software







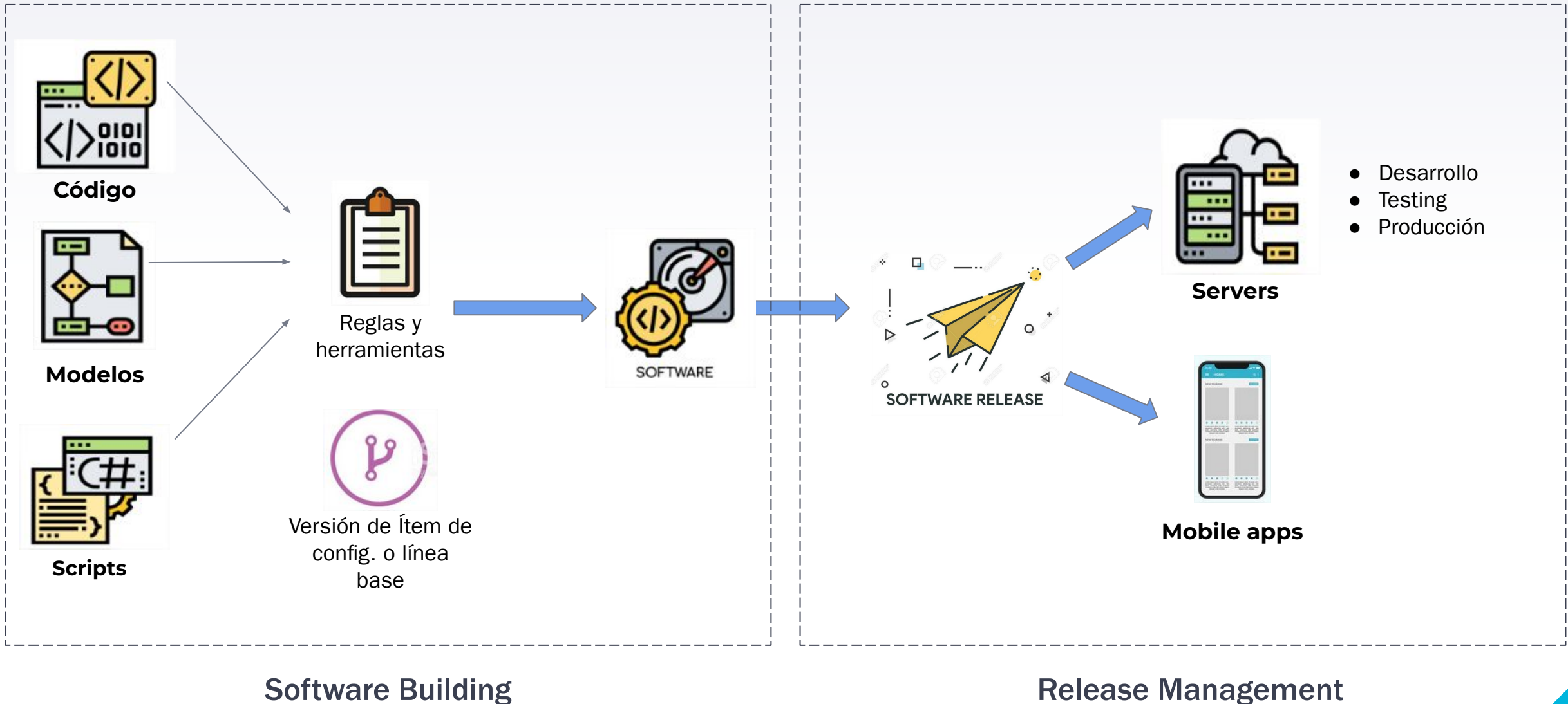
**Release Management** consiste en asegurar la construcción exitosa del paquete de software, basada en los ICs requeridos para la funcionalidad a entregar, para luego liberarlo en forma controlada a otros entornos ya sea de pruebas, producción, usuario final, etc.


Se divide en 2 partes:

- Software Building
- Release Management

Comprende también la administración, identificación y distribución de un producto Software

# Release Management del Software





**Software Building** es la construcción del paquete de software que será distribuido, utilizando las versiones correctas de los Ítems de Configuración y las herramientas apropiadas para construirlo.

# Conceptos clave de Software Building

## Build

- ▶ Un Build es el proceso de convertir archivos de código fuente en artefactos de software independientes que se pueden ejecutar en un entorno.

## Tipos de Build

- ▶ **Local builds:** El developer lo realiza localmente en su entorno de desarrollo, corre Pruebas Unitarias (UT)
- ▶ **Integration builds:** su objetivo es generar el entorno completo para pruebas de integración
- ▶ **Nightly builds:** su objetivo es ejecutar la construcción en forma diaria y generar reportes con información sobre estabilidad, tiempo de build, etc.
- ▶ **Release builds:** Se disparan cuando bien un administrador decide crear una nueva versión a ser liberada, o por el mismo sistema de integración si se utiliza el modo de deployment continuo

# Integración Continua

## Prácticas

- ▶ Disponer de un repositorio de código único.
- ▶ Automatizar el proceso de build mediante scripts o herramientas
- ▶ Hacer el build testeable automáticamente.
- ▶ Todo commit debe construirse por una herramienta de integración - no por el dev
- ▶ El build debe ser rápido
- ▶ Se prueba en un clon de producción
- ▶ Cualquiera debe obtener fácilmente la última versión del ejecutable
- ▶ Todos deben poder ver qué pasa con el proceso de integración (transparencia)
- ▶ Automatizar el proceso de despliegue (deployment)

## Responsabilidades del equipo


- ▶ Hacer check-in de código frecuentemente
- ▶ No subir código roto
- ▶ No subir código no testeado
- ▶ No subir código cuando el build no pasa
- ▶ No irse a casa hasta que el build central compile.



# Ventajas de la Integración Continua

Entre las ventajas de integración continua que más se suelen mencionar, se incluyen:

- ▶ Detección temprana y mejorada de errores, y métricas que le permiten abordar los errores a tiempo, a veces tras solo unos minutos de la incorporación
- ▶ Progreso continuo para mejorar el feedback
- ▶ Mejor colaboración en equipo; todos los miembros del equipo pueden cambiar el código, integrar el sistema y determinar rápidamente los conflictos con otras partes del software
- ▶ Integración mejorada del sistema, lo que reduce las sorpresas al final del ciclo de vida de desarrollo del software
- ▶ Menor número de errores durante las pruebas del sistema
- ▶ Sistemas actualizados constantemente en los que realizar las pruebas
- ▶ **Fin del “en mi máquina funciona”**



**Software Release Management** gestiona la identificación (Identification), ensamblado (packing) y la entrega (Delivery / Deployment) de los elementos de un producto Software (por ejemplo, un ejecutable, documentación, notas de lanzamiento o datos de configuración)

Define también el proceso de planificación, calendarización y gestión del software construido a lo largo de las etapas de desarrollo, testing, despliegue y soporte productivo.

# Definiciones Software Release Mgmt

## Versión

- ▶ Es una instancia de un sistema que es funcionalmente distinta en algún aspecto de otras instancias.

## Variante

- ▶ Una instancia de un sistema que es funcionalmente igual a otras instancias, pero difiere a niveles no funcionales (Ejemplo: Mismo producto para Version Linux / Version Windows)

## Release

- ▶ El término "Release" utilizado en este contexto, hace referencia a la distribución del Software fuera del entorno de desarrollo. En algunas herramientas, se utiliza el concepto de un "tag" como equivalente de release.

# Semantic Versioning

- ▶ Es una propuesta de un set de reglas y requerimientos que dictaminan como los números de versión son asignados e incrementados.

## Resumen de la propuesta

- ▶ Dado un número de versión **MAJOR.MINOR.PATCH**, se incrementa:
  - ▶ El número de versión **MAJOR** cuando se realizan cambios incompatibles en la API o SW
  - ▶ El número de versión **MINOR** cuando se agrega funcionalidad de una manera compatible con versiones previas
  - ▶ El número de versión **PATCH** cuando se arreglan bugs de forma compatible con versiones anteriores
- ▶ Se pueden agregar etiquetas adicionales para pre-releases o metadatos de build como extensiones al formato **MAJOR.MINOR.PATCH**.
- ▶ Ver <https://semver.org/> para más detalles de la propuesta

# Consideraciones para Release Mgmt

- ▶ Una vez generada la versión o release debemos buscar el mecanismo más efectivo para hacer despliegue -deploy- controlado a los distintos usuarios.
  - ▷ A estos mecanismos se los denomina Deployment Strategies
- ▶ Aspectos a evaluar para realizar un release:
  - ▷ 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

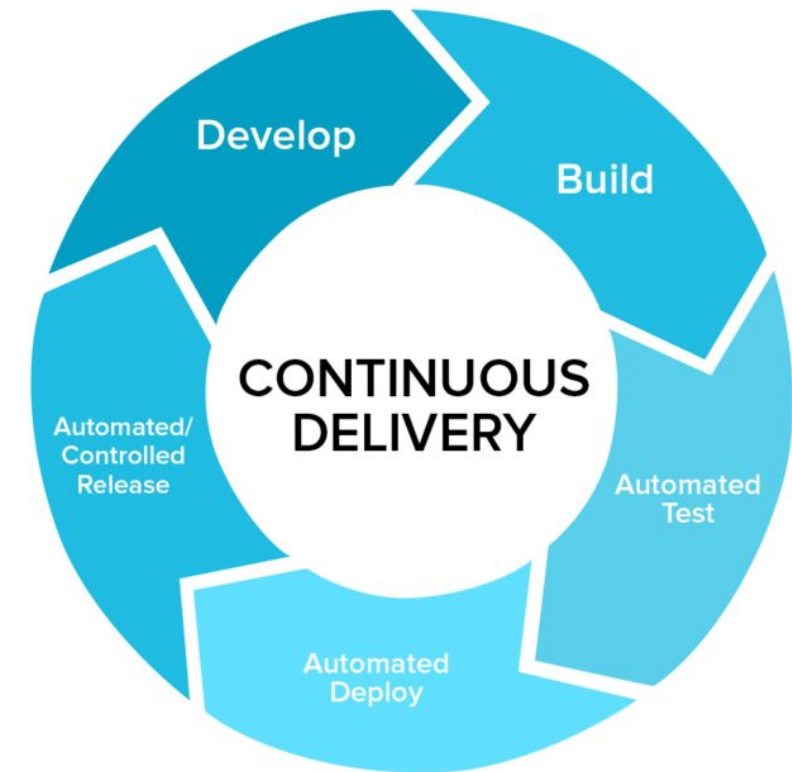


# Integración Continua

- ▶ La **integración continua (CI)** es una práctica de desarrollo de software mediante la cual los desarrolladores combinan los cambios en el código en un repositorio central de forma periódica, tras lo cual se ejecutan versiones y pruebas automáticas.
- ▶ La integración continua La integración continua se refiere en su mayoría a la fase de creación o integración del proceso de publicación de software y conlleva un componente de automatización (p. ej., CI o servicio de versiones) y un componente cultural (p. ej., aprender a integrar con frecuencia)
- ▶ Es una práctica requerida para poder realizar entregas continuas (Continuous Delivery)

# Continuous Delivery

- ▶ La **Entrega Continua** es un conjunto de prácticas que permiten asegurar que el software puede desplegado en producción rápidamente y en cualquier momento, aplicando el concepto de Pipelines
- ▶ Los artefactos resultantes del proceso de construcción incluyen código ejecutable y bibliotecas, que luego se puede combinar en una instalación paquete e implementado en un entorno para verificación o uso de producción.
- ▶ Esto lo logra haciendo que cada cambio llegue a un entorno de staging o semi productivo, donde se corren exhaustivas pruebas de sistema. Luego se puede pasar a producción manualmente cuando alguien apruebe el cambio con solo apretar un botón.



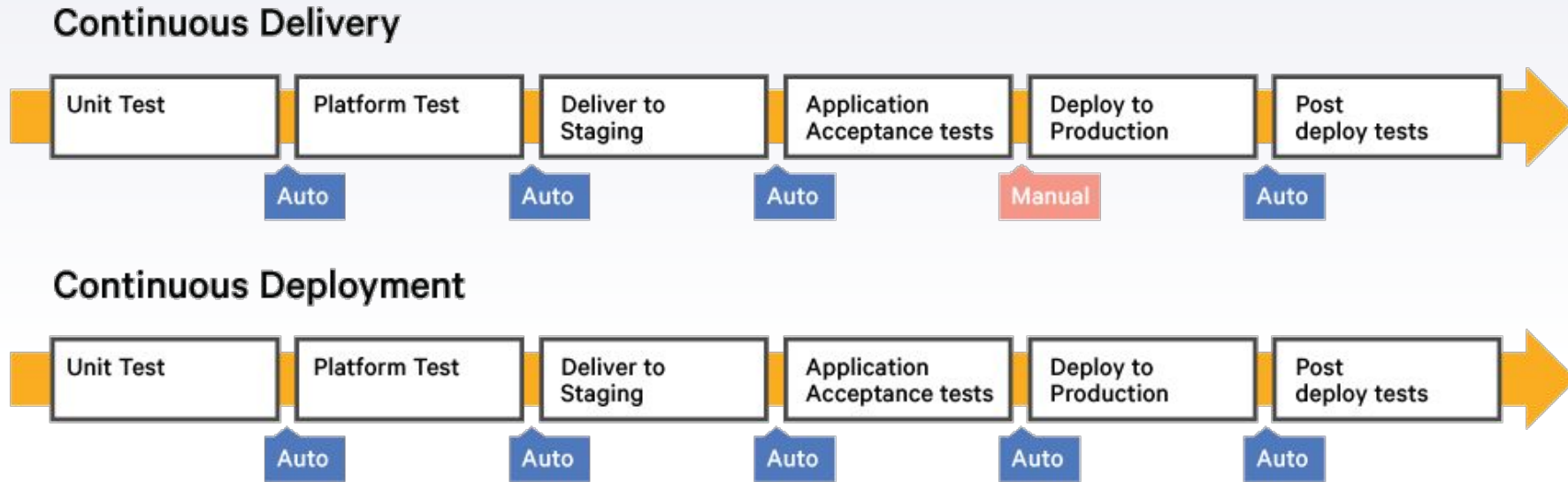
# Pipelines en Continuous Delivery

Para poder cumplir con el objetivo, los procesos de Continuous Delivery establecen un pipeline de acciones para poder construir e instalar el producto Software.

Usualmente estos Pipelines suelen contener las siguientes etapas:

- ▶ Testing and QA del Software Construido
  - ▷ Ejecuta las pruebas automáticas
- ▶ Change Management, Governance, and Approvals
  - ▷ Contiene las actividades de aprobación de los cambios a desplegar
- ▶ Deployment Strategies
  - ▷ Define la estrategia con la que el Software será desplegado en el ambiente
- ▶ Verification
  - ▷ Define las actividades para verificar que el software fue correctamente desplegado
- ▶ Rollback
  - ▷ Define las acciones para volver al estado anterior, en el caso de que falle la verificación

# CD vs CD



El **Continuous deployment** es el paso siguiente al **Continuous Delivery**, en donde también el despliegue a producción se realiza en forma automática por un proceso y no por personas.

# Diferencias entre integración continua (CI), entrega continua (CD) y despliegue continuo

La **integración continua** es un proceso de desarrollo de software en el que los desarrolladores integran el código nuevo que han escrito con una mayor frecuencia a lo largo del ciclo de desarrollo, y lo añaden a la base de código al menos una vez al día.

Se realizan pruebas automáticas en cada iteración de la compilación para identificar los problemas de integración con más rapidez, cuando son más fáciles de solucionar, para así evitar problemas en la construcción final para la publicación.

En el **despliegue continuo**, los cambios de código que se realizan en las aplicaciones se publican automáticamente en el entorno de producción. Esta automatización se basa en una serie de pruebas predefinidas. Una vez que las nuevas actualizaciones pasan esas pruebas, el sistema envía las actualizaciones directamente a los usuarios del software.

La **entrega continua (CD)** retoma el proceso donde finaliza la integración continua y automatiza la entrega de aplicaciones a los entornos de infraestructura seleccionados.

La CD se centra en entregar cualquier cambio validado de la base de código (actualizaciones, correcciones de errores e incluso nuevas características) a los usuarios de la forma más rápida y segura posible. Garantiza la automatización del envío de los cambios del código a los diferentes entornos, como desarrollo, pruebas y producción.



# Deployment Strategies

- ▶ Un deployment strategy es un mecanismo de cambiar o actualizar una aplicación de Software.
- ▶ El objetivo es realizar el cambio sin generar un downtime, de forma tal que los usuarios finales no noten el cambio de la aplicación.
- ▶ Existen varias estrategias de despliegue que revisaremos a continuación



# Basic Deployment

## ▶ Concepto

- ▶ Todos los nodos de la aplicación son actualizados en el mismo momento con la nueva versión

## ▶ Ventajas

- ▶ Muy simple de utilizar, no hay mecanismos específicos que debamos realizar para este deploy

## ▶ Desventajas

- ▶ Riesgo alto de downtime



# Blue / Green Deployment

## ► Concepto

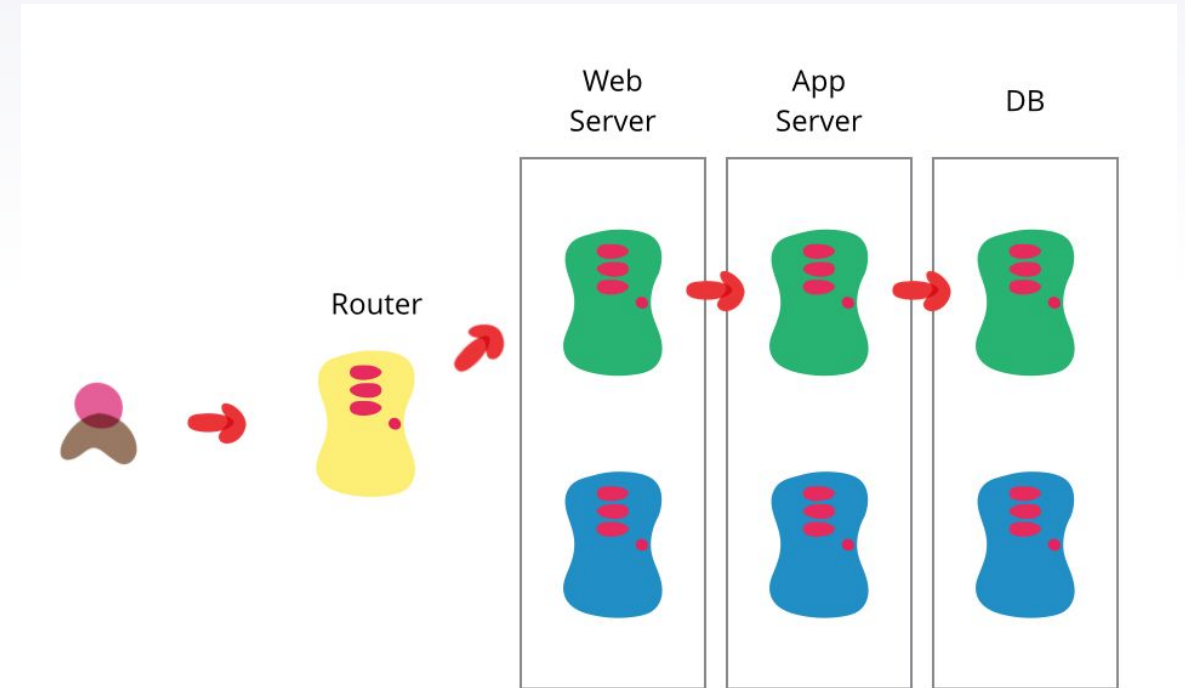
- Disponemos de dos ambientes lo más similar posible, uno "green" y el otro "blue"
- El despliegue se realiza sobre uno de los ambientes, mientras el otro contiene la versión anterior
- Un balanceador selecciona a qué grupo de servers utilizar como productivo

## ► Ventajas

- Simple de implementar

## ► Desventajas

- En organizaciones grandes, se torna una solución costosa



# Rolling Deployment

## ► Concepto

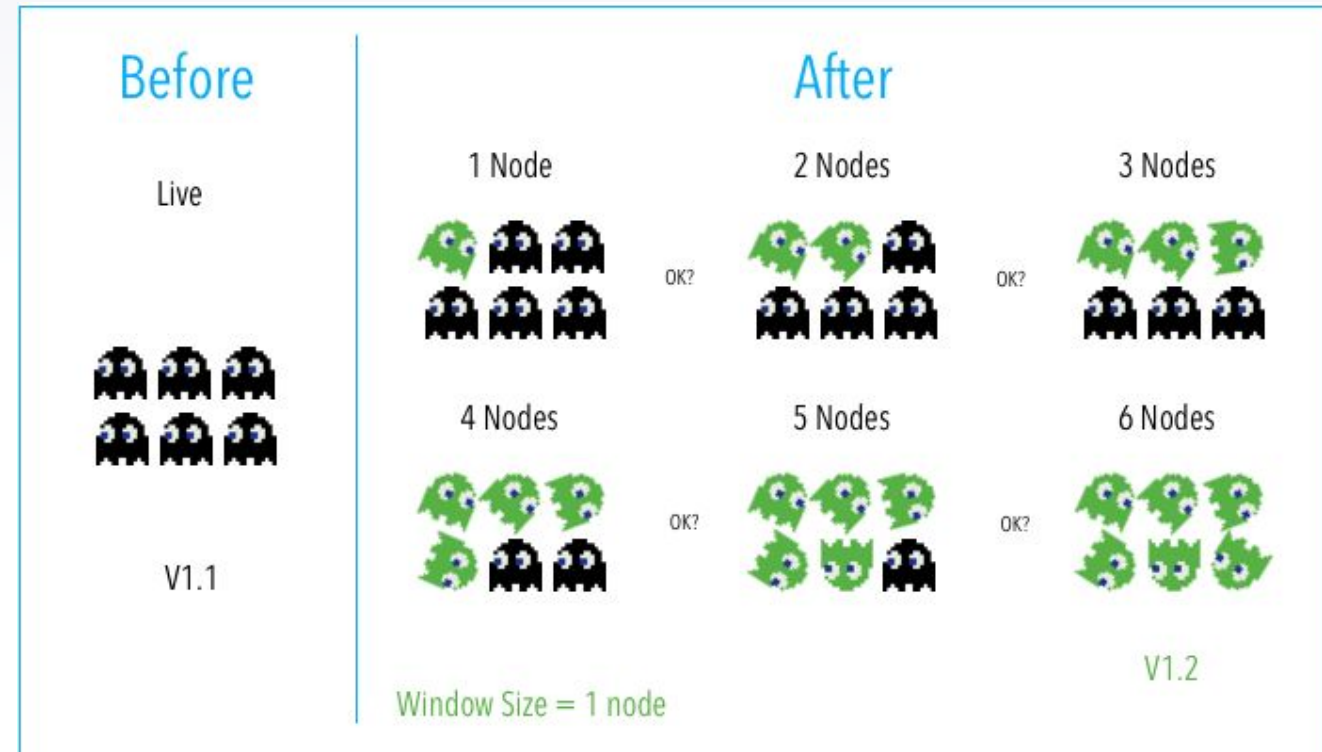
- Los nodos de un ambiente son actualizados 1 a 1 (o en lotes) con la nueva versión del software.

## ► Ventajas

- Rollout gradual con un incremento de tráfico controlado. Simple de hacer rollback

## ► Desventajas

- App / BD necesitan mantener consistencia para trabajar con versiones distintas (+1 y -1 en cada pod)



# Canary Deployment

## ► Concepto

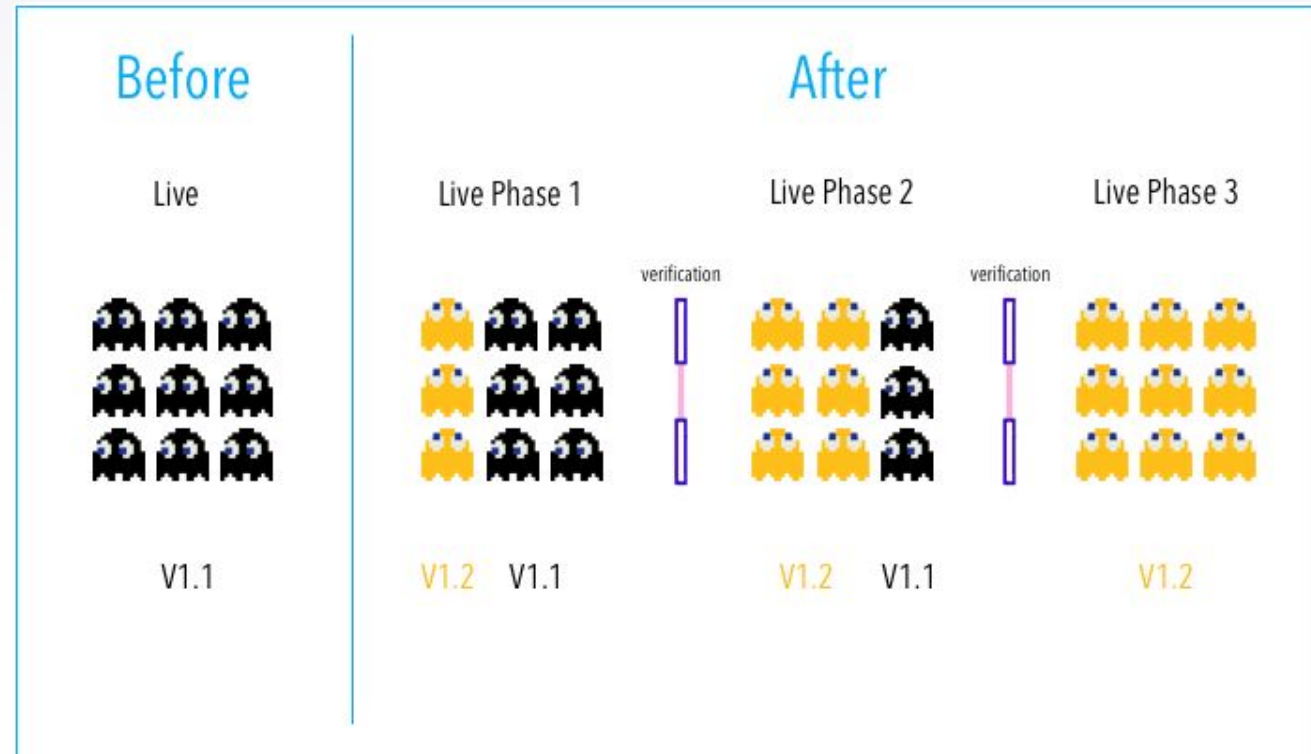
- Conceptualmente consiste en desplegar el cambio en un set controlado de nodos, e ir chequeando el comportamiento
- Es actualmente una de las formas más comunes de hacer deploy

## ► Ventajas

- El despliegue es en pequeñas fases (por ejemplo 2% de los nodos, 10%, 50%, 75%, 100%)
- En el caso de fallas, se quitan los nodos que fueron incorporándose
- Solo existe 1 ambiente de producción

## ► Desventajas

- Es complejo de testear producción.
- Herramientas de monitoreo complejas para entender que está desplegado



# Release Management Metrics

- ▶ Métricas asociadas a los flujos de building y de releasing del software





# Métricas relacionadas a deployment

## Deployment Frequency

Cada cuanto tiempo una organización despliega software a Producción satisfactoriamente

- Medida: Cant. Deployments por día
- ¿Para qué? : Indica cada cuanto tiempo agregamos valor a los clientes

## Lead Time for Changes

El tiempo que le lleva a un Commit llegar a Producción

- Medida: Lead Time en días
- ¿Para qué? : Potencialmente puede indicar la productividad del proceso de desarrollo

## Change Failure Rate

El porcentaje de despliegues que llevaron a una degradación del servicio

- Medida: Número de Incidentes / Número de despliegues
- ¿Para qué? : Incrementar la satisfacción de los clientes al reducir el número de downtimes

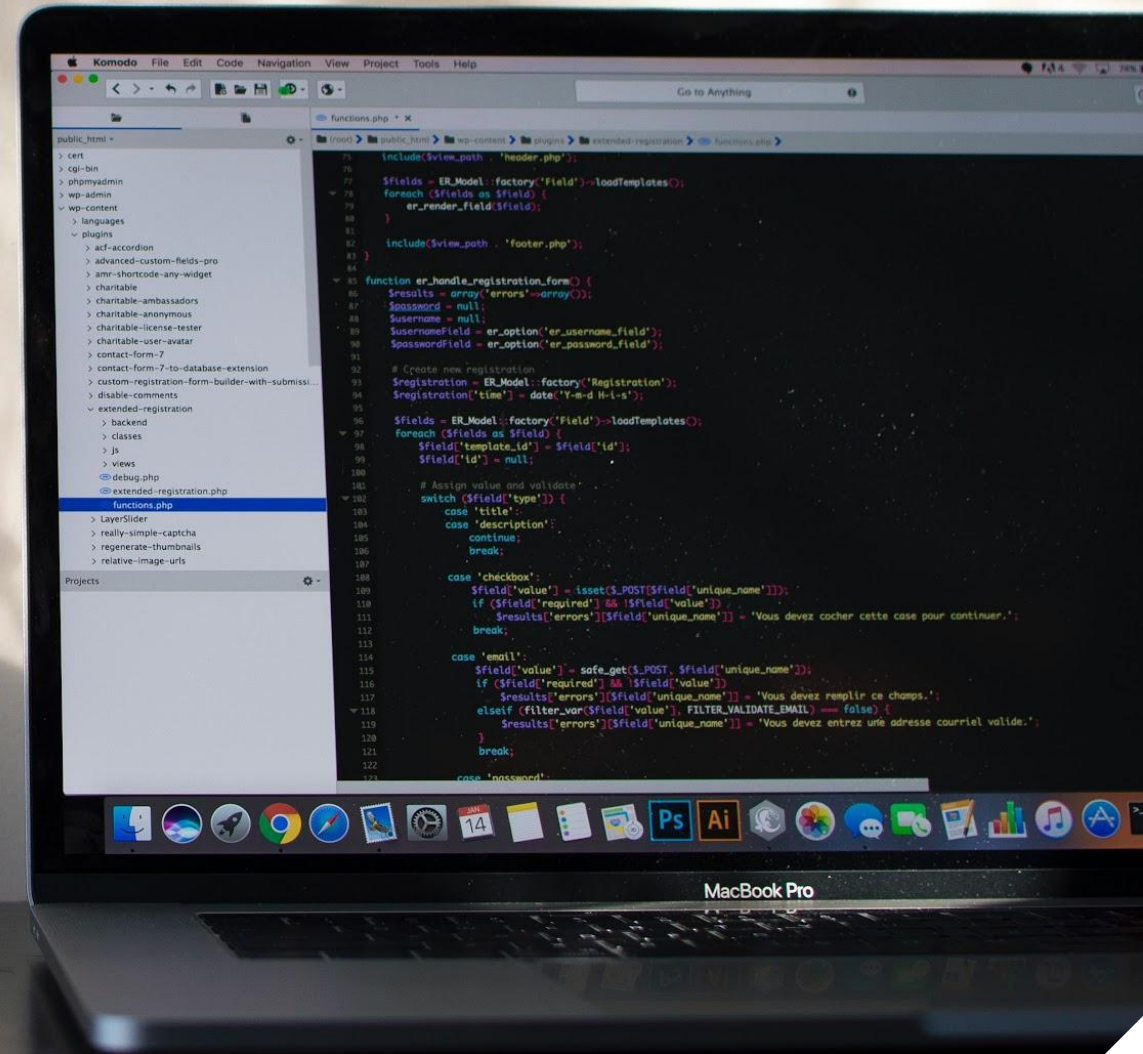
## Mean Time To Recovery (MTTR)

El tiempo que le demora a la organización recuperarse luego de un incidente

- Medida: Tiempo medido en horas
- Para qué? : Incrementar la satisfacción de los clientes al reducir el número de downtimes

# Software Configuration Management Tools

## Una vista Landscape

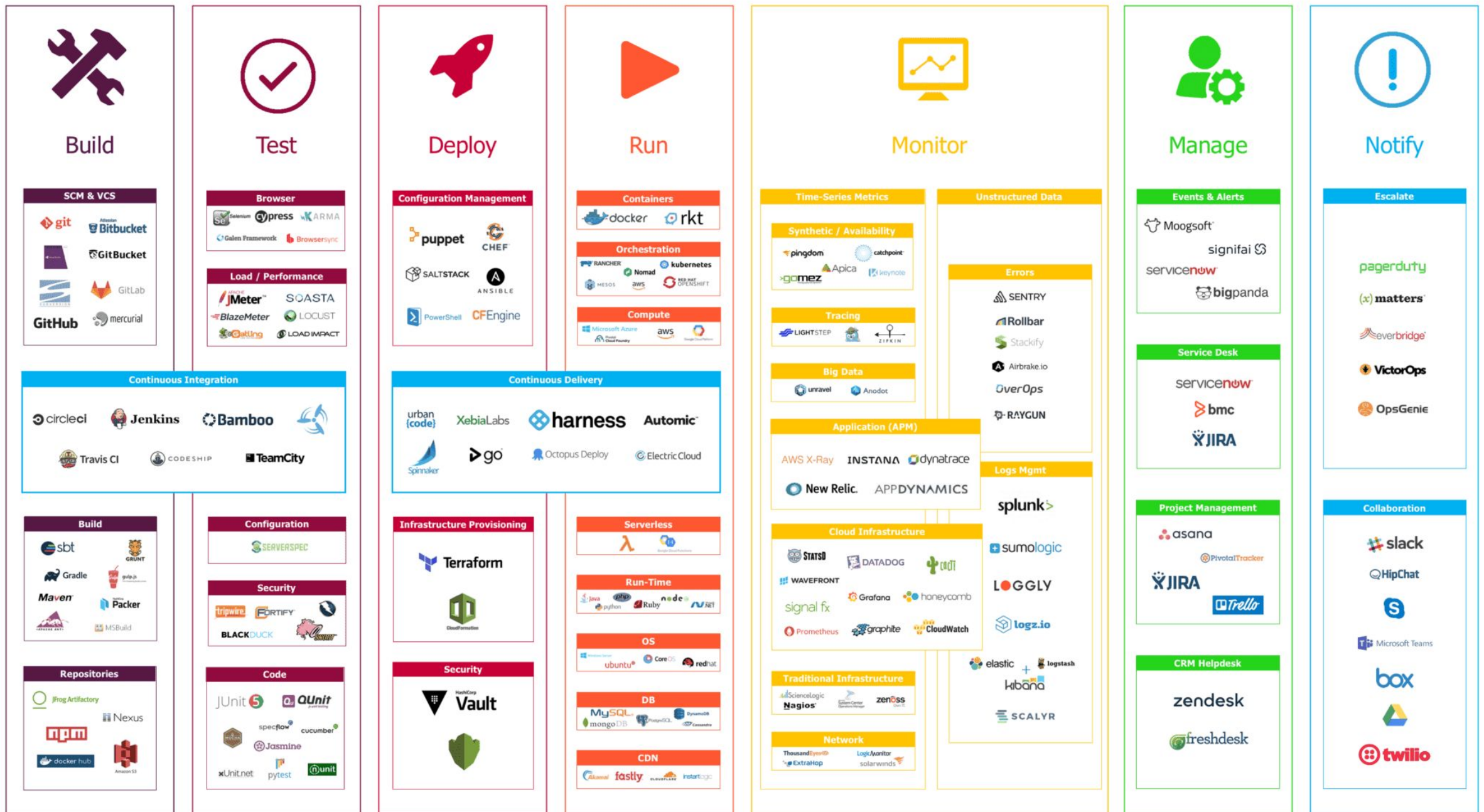


# SCM Tooling

Las herramientas dan soporte a la gestión de configuración en diferentes niveles. Cada organización define qué herramientas utiliza en esta gestión.

Dentro de las herramientas, podemos mencionar:

- ▶ Herramientas que gestionan las versiones de los código fuente, archivos de configuración y artefactos relacionados.
- ▶ Herramientas que ayudan en la automatización del Build y establecen mecanismos (Vía Pipelines) para permitir el delivery continuo
- ▶ Repositorios que almacenan las historias de los builds
- ▶ Herramientas que gestionan los cambios funcionales y el control de cambios
- ▶ Herramientas que dan soporte al despliegue de aplicaciones
- ▶ Herramientas colaborativas para comunicar





# Toolings de automatización de entornos

## Metal / Físico



**Hardware:** Dependiente (drivers de placas, video, controladores, etc.)

**Aislamiento:** Todos los servicios comparten el SO y kernel pudiendo afectar a otros.

**Cambios:** No hay snapshots, al romperse hay que reinstalar.

**Performance:** No hay overhead de performance

**Aprovisionamiento:** Requiere scripts con herramientas para gestión de paquetes, dependencias, acceso al servidor físico.

## Máquina Virtual



**Hardware:** Cuasi independiente — hardware virtualizado, solo respeta arquitectura (x86, ARM,..)

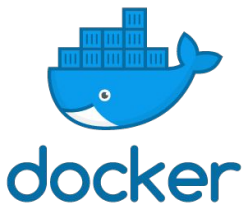
**Aislamiento:** Todos los servicios comparten el SO y el kernel pudiendo afectar a otros.

**Cambios:** Más fáciles que *metal*, existen snapshots, y por ende se puede versionar una VM.

**Performance:** Hay una gran pérdida de performance ya que cada máquina tiene varios S.O.

**Aprovisionamiento:** Al tener las imágenes de base y snapshots de una máquina virtual, el aprovisionamiento es más rápido y simple. Un developer podría tener su entorno similar a producción.

## Contenedores



**Hardware:** Independiente del hardware — solo respeta arquitectura. Memoria/Cpu flexible.

**Aislamiento:** Filosofía de *1 container por servicio*, aislando los servicios entre sí.

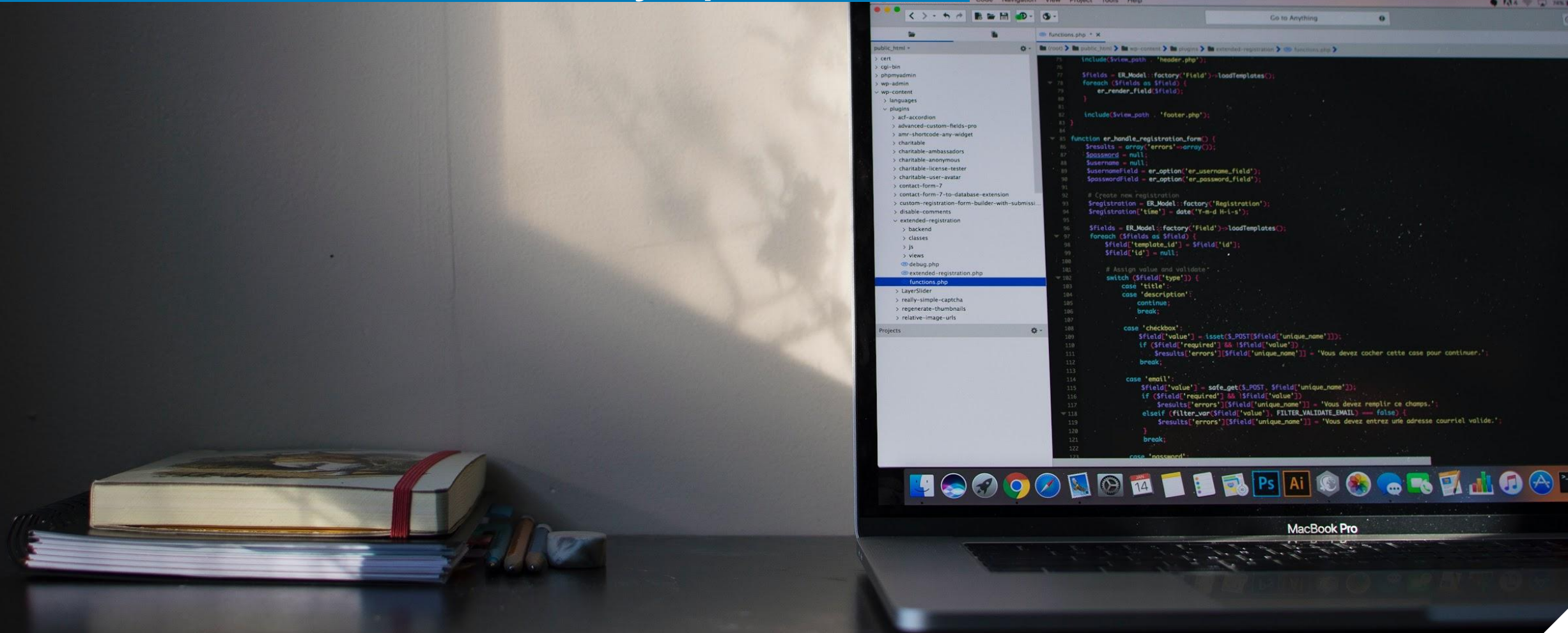
**Cambios:** Tienen archivos de configuración, son versionables, hay repositorios de containers.

**Performance:** Mucho más livianos que VM, pierde muy poca performance que metal.

**Aprovisionamiento:** Utiliza capas de aprovisionamiento, soporta snapshots, el bootup es en segundos o minutos. Fácil de mover cambios desde y hacia producción.

# Devops

## La fusión entre desarrollo y operaciones





# El problema a resolver

- ▶ En organizaciones de las llamadas tradicionales
  - ▷ El que desarrolla no escribe los requerimientos
  - ▷ El que desarrolla no hace testing funcional
  - ▷ El que desarrolla no pasa a producción
  - ▷ El que desarrolla no está de guardia

Durante décadas los operadores de IT y sistemas vivieron en plena queja con desarrolladores que no consideraban

- A. que los entornos pueden tener diferencias
- B. que la puesta en producción no es gratuita
- C. cuestiones como la escalabilidad, performance, disponibilidad, tolerancia a fallos, entre otros.
- D. hacer software no es fire & forget™, y esperaban que operaciones maneje automáticamente cualquier problema, sin conocimiento del dominio



**GENERANDO FALTA DE COMUNICACIÓN Y COLABORACIÓN**

# El problema a resolver



t Owner) requiere de  
e: **Nuevos Riesgos!**  
**Mantengamos la estabilidad!**  
a "Deployando" una  
de la pared" a

os de Deploy y  
mplementación.  
ite los scripts de  
s por los  
os scripts.

**DevOps** es un 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 también un cambio cultural, ya que en un modelo de DevOps, los equipos de desarrollo y operaciones ya no están “aislados”. A veces, los dos equipos se fusionan en uno solo, donde los ingenieros trabajan en todo el ciclo de vida de la aplicación, desde el desarrollo y las pruebas hasta la implementación y las operaciones, y desarrollan una variedad de habilidades no limitadas a una única función.

**DevOps NO es una metodología**, es más bien un conjunto de prácticas, herramientas y una filosofía cultural que automatizan e integran los procesos entre los equipos de desarrollo de software y IT.

El término se acuñó en una conferencia de infraestructura en Bélgica que quería atraer a desarrolladores

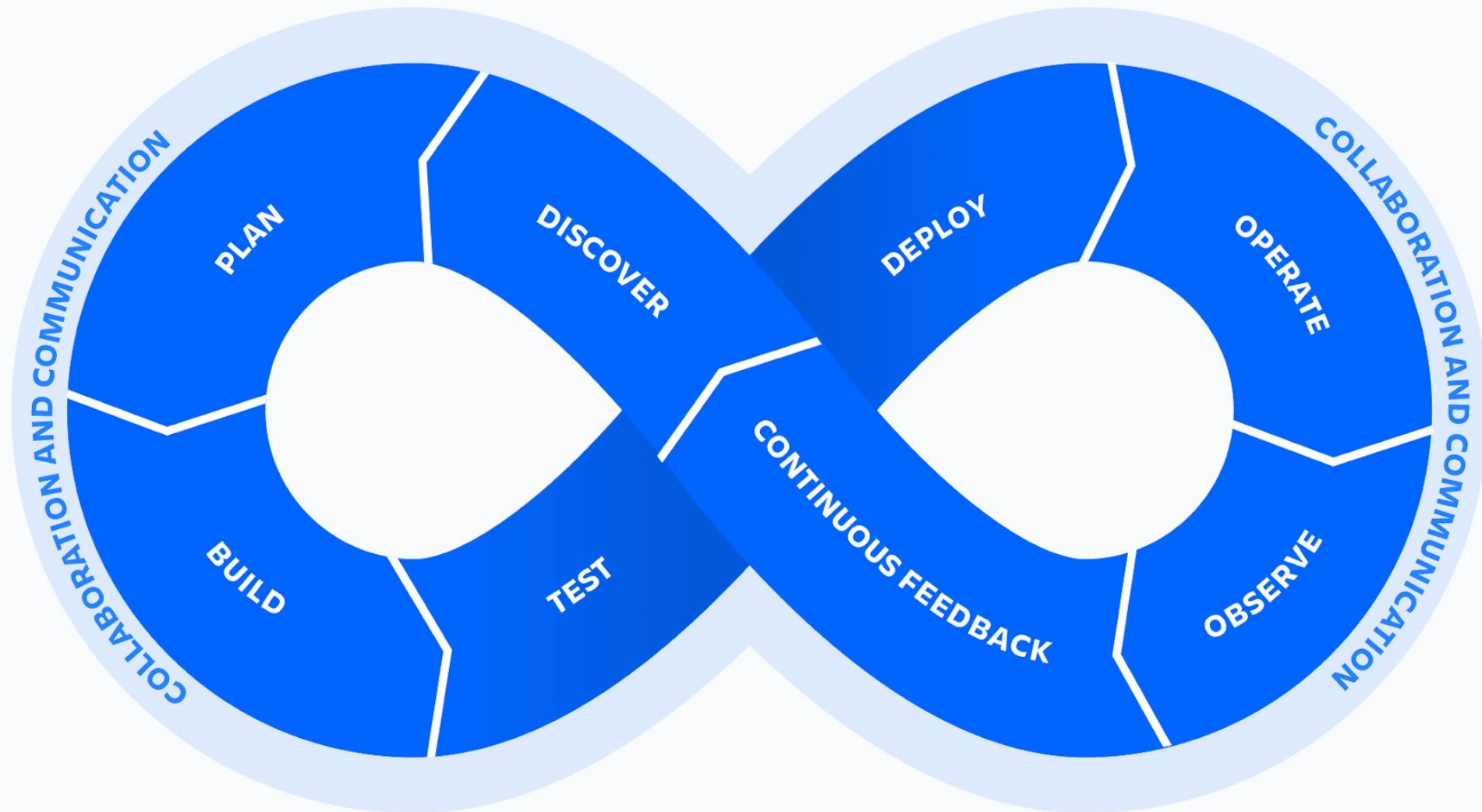
# DevOps: CALMS

**CALMS** es la filosofía de trabajo detrás de DevOps, dónde:

- ▶ **Cultura**: Ser dueños del cambio para mejorar la colaboración y comunicación.
- ▶ **Automatización**: Eliminar el trabajo manual y repetitivo lleva a procesos repetibles y sistemas confiables, reduciendo 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 fallas para que otros puedan aprender.



# El ciclo de vida DevOps



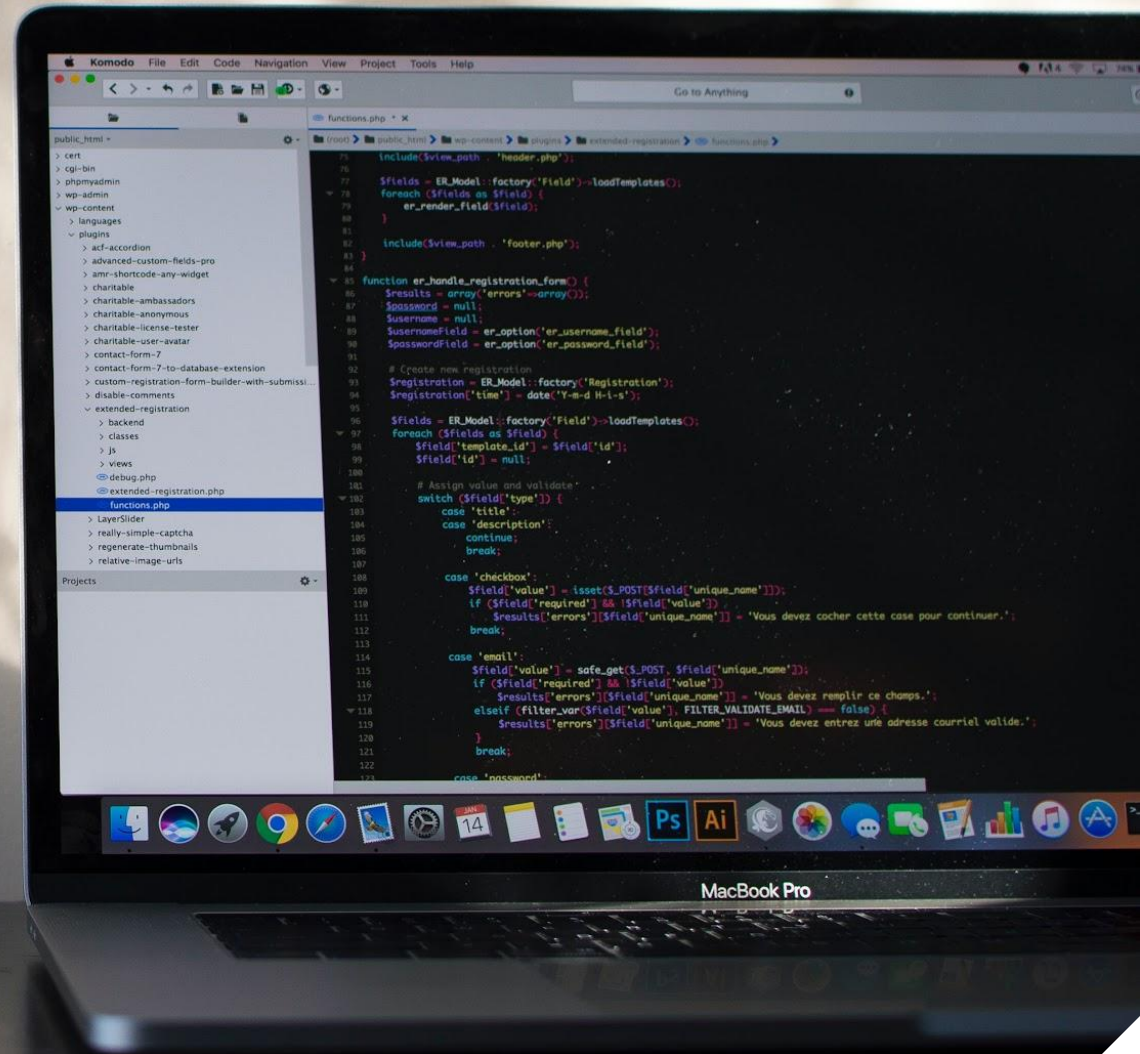
# Prácticas de DevOps en cada etapa

- ▶ 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



# Escenario: SCM en la vida Real

## Aplicación práctica de la disciplina



# SCM en la Vida Real (Práctica)

Una compañía los contrató porque todo su equipo de desarrollo renunció y necesitan terminar un proyecto de un cliente.

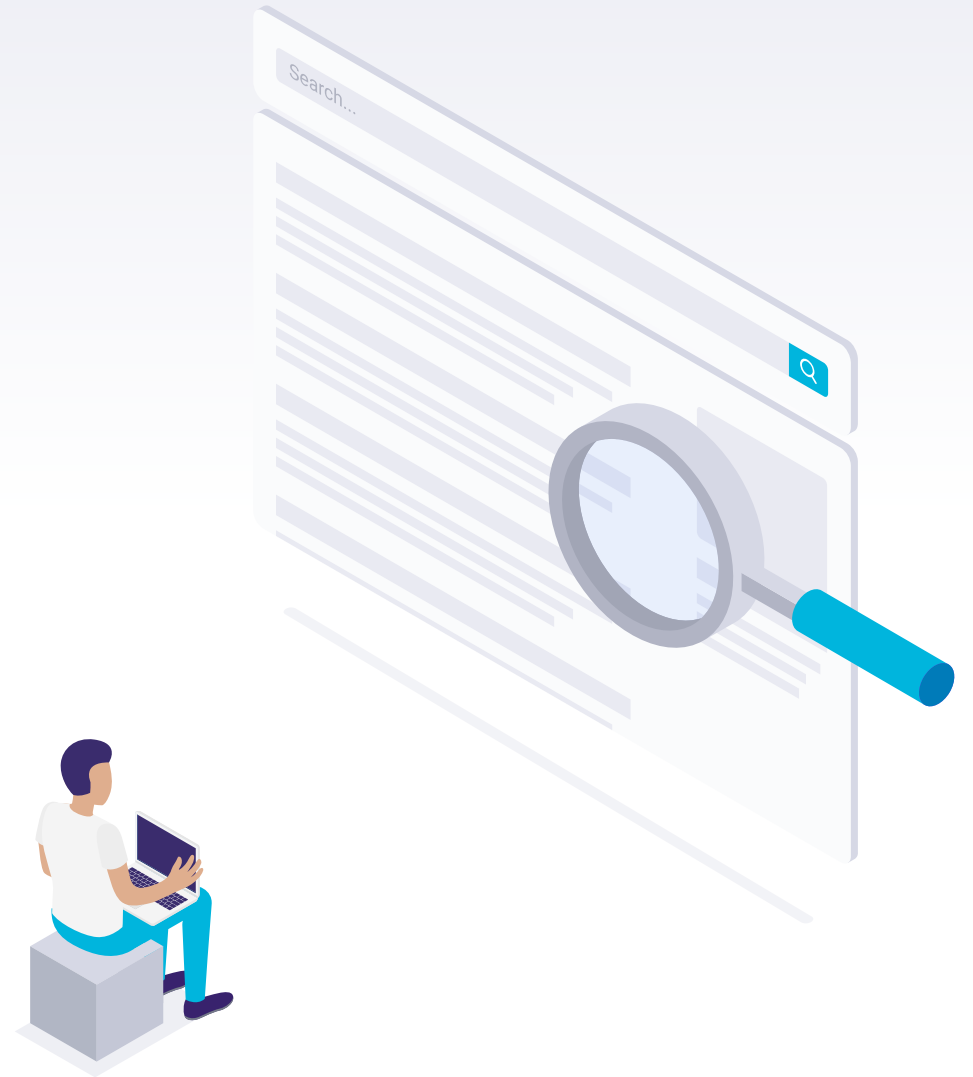
Solo les dieron el ordenador del líder técnico y un par de documentos.

- ¿Cómo podemos aplicar los conceptos en este caso?
- ¿Cómo se asocian las actividades de SCM en este escenario?

# ¡Gracias!

## ¿Alguna pregunta?

Ingeniería en Software - UTN FRBA  
Unidad SCM



# Bibliografía recomendada

## ► Referencias

- SWEBOK – Capítulo 7 – Software Configuration Management

## ► Libros

