

TECNOLOGÍAS DE LA INFORMACIÓN EN EL DESARROLLO DE SOFTWARE MULTIPLATAFORMA

08 de Octubre del 2020

CONTROL DE VERSIONES

Equipo NERF – 4 cuatrimestre Sección B

INTEGRANTES:

19170038 – Raúl Alejandro De los santos Anguiano

19170106 – José Felipe Muñiz Fonseca

19170068 – Néstor Josué Puentes Inchaurregui

DOCENTE: MSC. Sosa Escobedo Abel



INDICE



CONCEPTOS BÁSICOS SOBRE EL CONTROL DE VERSIONES

*Trunk *Branch *Build *Revision *Estrategias de versionamiento
*Stable Trunk/Stable Branch *Unstable Trunk/Unstable Branch
*Agile *Versionamiento *Testing

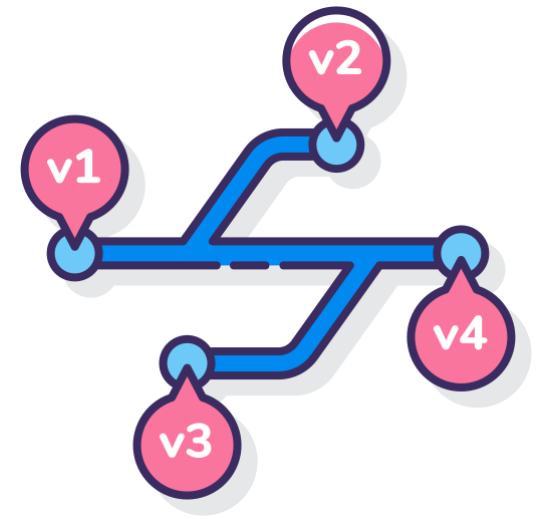


VIDEO COMPLEMENTARIO.



CONCLUSIONES.

INTRODUCCION



Los controladores de versión son herramientas muy poderosas para cualquier proyecto en desarrollo que involucre cambios constantemente, como edición de fotos, videos, paginas web, desarrollo de software, entre otras más.

Esta herramienta, actúa como si fuera un diario, en el cual permite detallar y almacenar los cambios que fueron realizados; indica que líneas o que archivos fueron editados, cuando fueron editados y por quien fueron editados. Es tan poderosa que permite “crear” copias exactas de tu proyecto, mejor conocido como clonar tu repositorio que es el lugar donde almacenas todas las carpetas con sus archivos de dicho proyecto y editar digamos tu código sobre esa copia y hacer pruebas para que en caso de que ocurra algún error, tu proteger tu proyecto oficial este protegido.

An abstract network diagram with various sized nodes (black, blue, and grey) connected by thin grey lines. Some nodes are highlighted with larger concentric circles. The background is light grey with faint, larger-scale network patterns.

CONTROL DE VERSIONAMIENTO

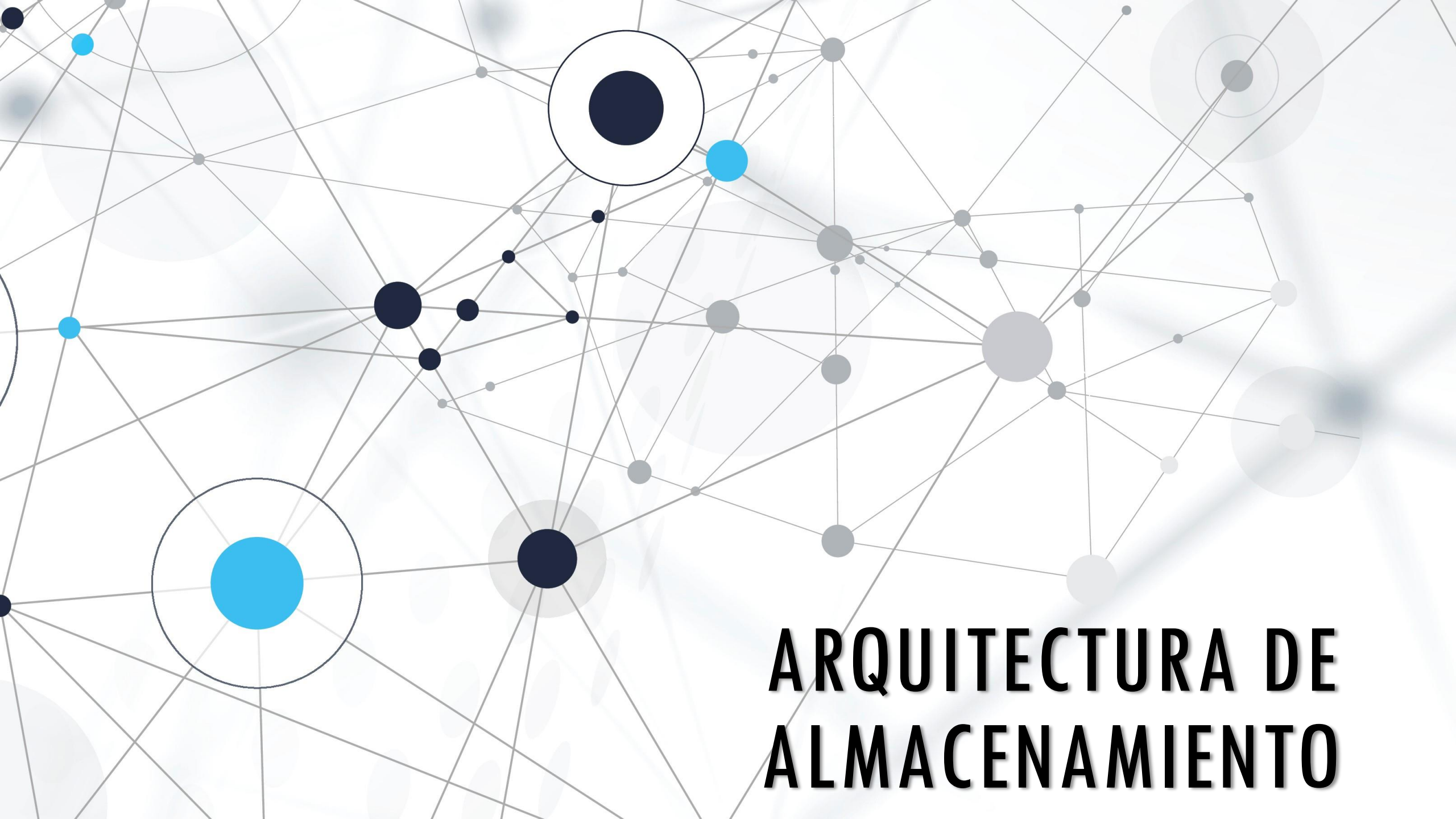
CONTROL DE VERSIONES O ADMINISTRACIÓN DE LA CONFIGURACIÓN DEL SOFTWARE

“Se llama control de versiones a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo. Una versión, revisión o edición de un producto, es el estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación”

Características

Un sistema de control de versiones debe proporcionar:

- Mecanismo de almacenamiento de los elementos que deba gestionar (ej. archivos de texto, imágenes, documentación...).
- Posibilidad de realizar cambios sobre los elementos almacenados (ej. modificaciones parciales, añadir, borrar, renombrar o mover elementos).
- Registro histórico de las acciones realizadas con cada elemento o conjunto de elementos (normalmente pudiendo volver o extraer un estado anterior del producto).



ARQUITECTURA DE ALMACENAMIENTO

ARQUITECTURAS DE ALMACENAMIENTO

*Distribuidos: cada usuario tiene su propio repositorio. Los distintos repositorios pueden intercambiar y mezclar revisiones entre ellos. Es frecuente el uso de un repositorio, que está normalmente disponible, que sirve de punto de sincronización de los distintos repositorios locales. Ejemplos: Git y Mercurial.

*Centralizados: existe un repositorio centralizado de todo el código, del cual es responsable un único usuario (o conjunto de ellos). Se facilitan las tareas administrativas a cambio de reducir flexibilidad, pues todas las decisiones fuertes (como crear una nueva rama) necesitan la aprobación del responsable. Algunos ejemplos son CVS, Subversion o Team Foundation Server.





CONCEPTOS BASICOS

CONCEPTOS BASICOS

El repositorio

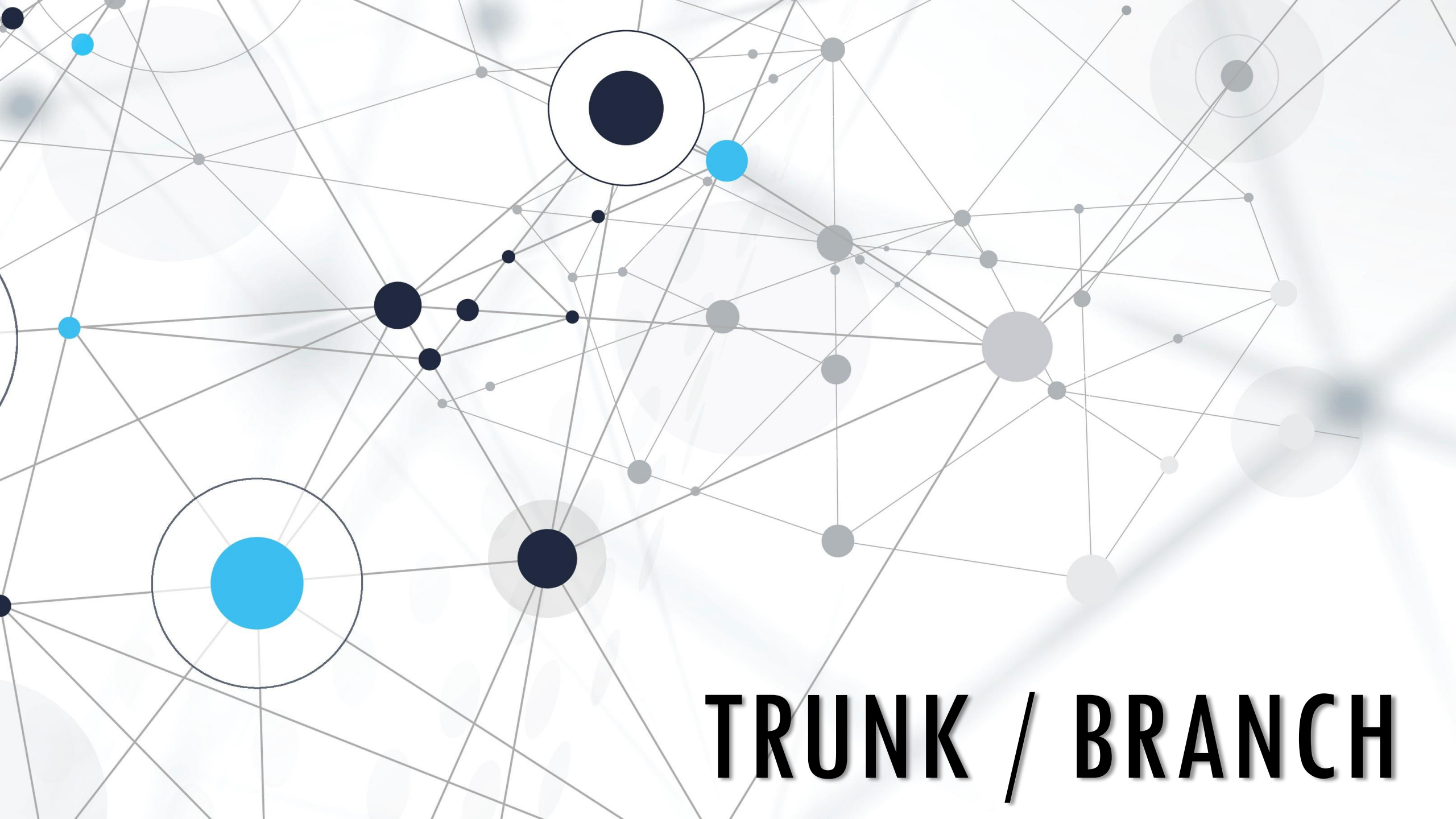
Subversion usa una base de datos central que contiene todos los archivos cuyas versiones se controlan y sus respectivas historias. Ésta base de datos se conoce como el repositorio. El repositorio normalmente yace en un servidor de archivos ejecutando el servidor de Subversion, que provee a pedido el contenido a los clientes de Subversion (como TortoiseSVN, por ejemplo). Si solo puede hacer una copia de seguridad de una sola cosa, hágala del repositorio, ya que es la copia maestra de toda su información.



Copia de trabajo

Acá es donde se realiza el trabajo en serio. Cada desarrollador tiene su propia copia de trabajo, comúnmente conocida como caja de arena en su computador local. Usted puede obtener la última versión del repositorio, trabajar en ella localmente sin perjudicar a nadie, y cuando esté feliz con los cambios que ha realizado puede confirmar sus cambios en el repositorio.





TRUNK / BRANCH

TRUNK (TRONCO)

* **Versión base de un proyecto.**

Es la línea base o **rama principal de desarrollo**. Sobre esta rama se tiene el código que ya está preparado para una **lanzamiento**. Idealmente debería tener un control de pruebas, test,... que permita que este código esté en un estado estable.

BRANCH (RAMA)

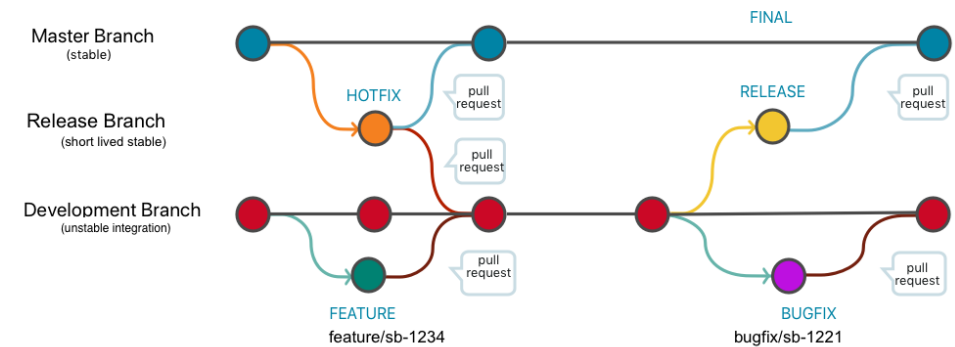
* Es una copia de código para hacer cambios. Lo ideal es que **cada desarrollador trabajara en su rama** y luego, una vez validado los cambios, hiciera la **integración a su trunk**.

Se utiliza para mantener una versión especial del proyecto para un propósito específico y fijado de antemano.

Por ejemplo, desarrollar una nueva funcionalidad sin contaminar la rama superior con modificaciones no validadas.

Cuando la rama es estable (errores solucionados, funcionalidades añadidas) se fusiona (merge) con el tronco.

Copyright © 2018 Build Azure LLC





BUILD

BUILD (CONSTRUIR)

Build, hace referencia a inicializar un proyecto, es decir, crear un repositorio.

Un repositorio es, tu proyecto guardado en algún sistema de control de versiones, ahí se guardan todas tus carpetas tal cual las tienes creadas de manera local. Y cuando tienes un proyecto creado (*un repositorio*) puedes hacer uso de clonar ese mismo proyecto para que no se vea afectado el proyecto principal.





REVISION

REVISION

Revision Control System o **RCS** es una implementación en software del control de versiones que automatiza las tareas de guardar, recuperar, registrar, identificar y mezclar versiones de archivos.

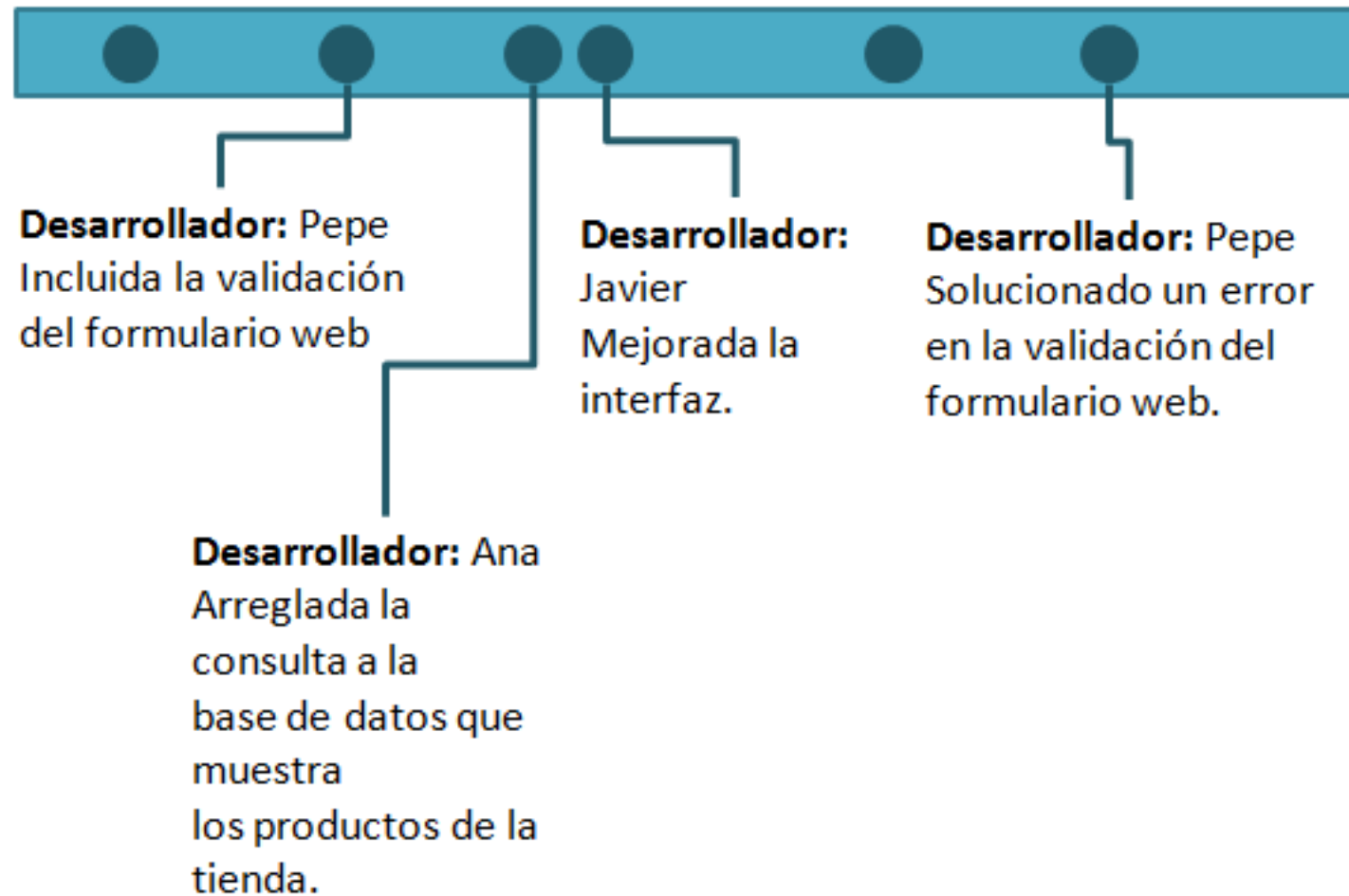
RCS es útil para archivos que son modificados frecuentemente, por ejemplo programas informáticos, documentación, gráficos de procedimientos, monografías y cartas.



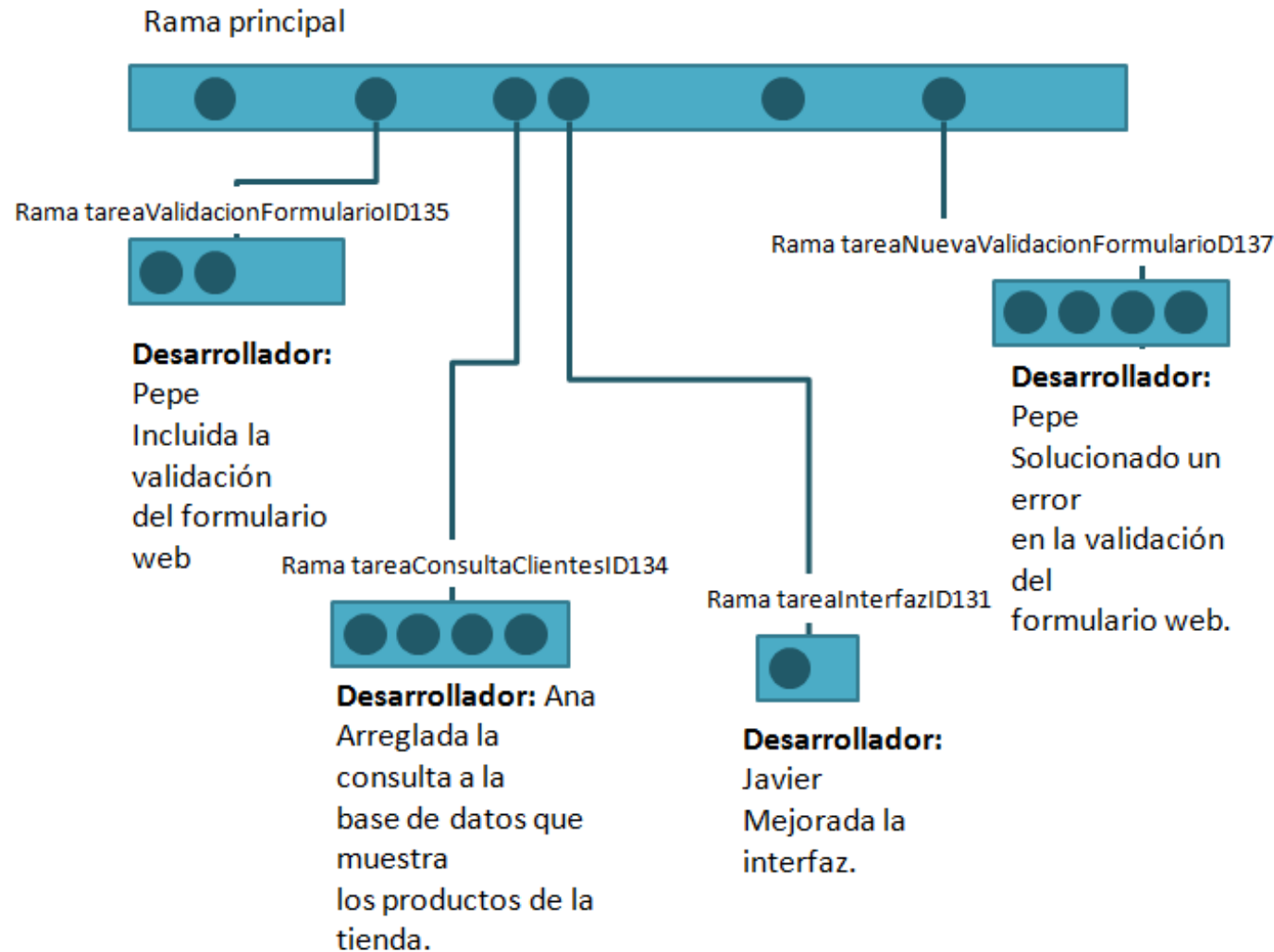
An abstract network diagram with various sized nodes (circles) in dark blue, light blue, and grey, connected by thin grey lines. Some nodes are highlighted with larger concentric circles. The background is white with faint, larger-scale network patterns.

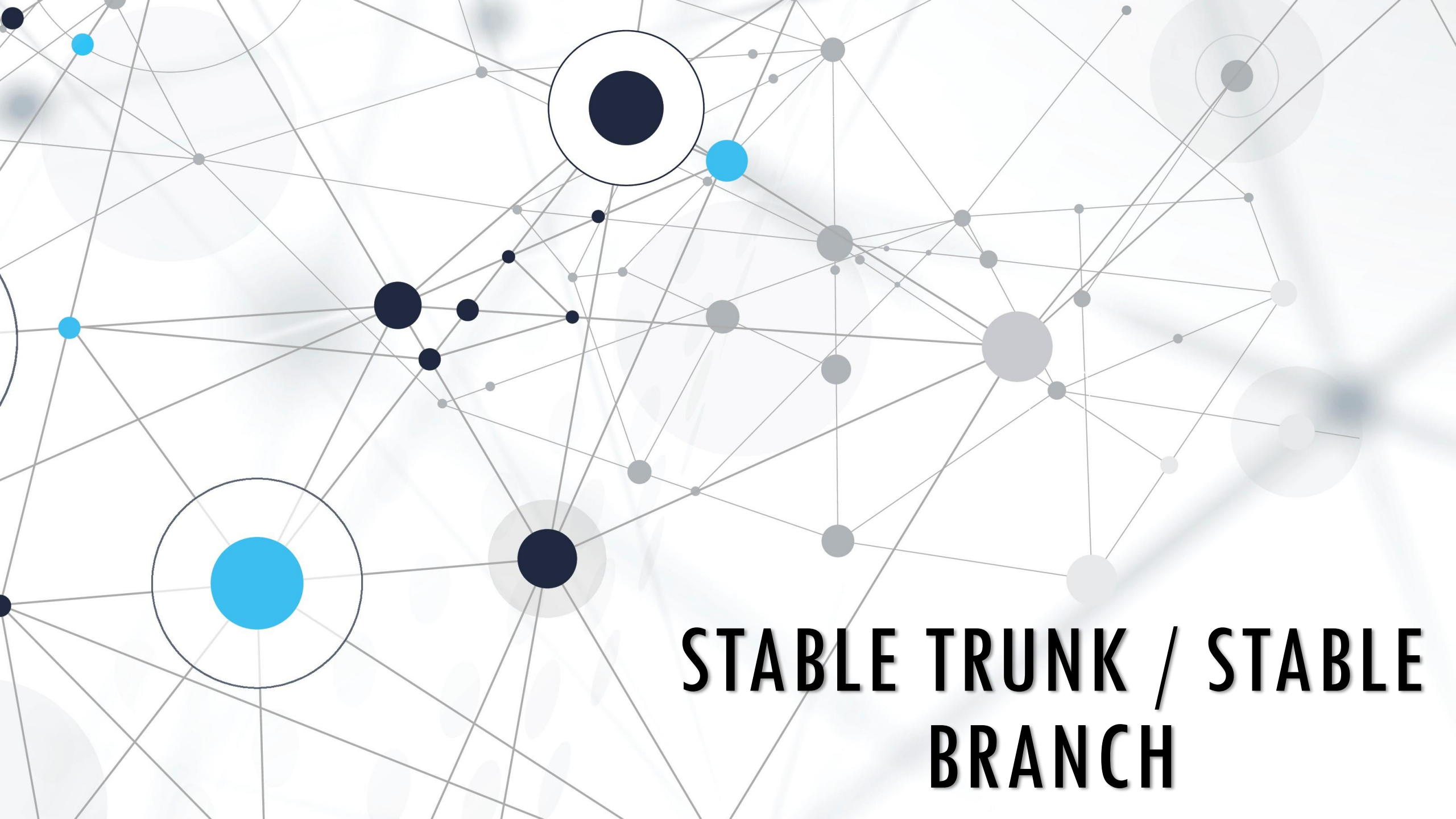
ESTRATEGIAS DE VERSIONAMIENTO

ESTRATEGIAS DE VERSIONAMIENTO



ESTRATEGIAS DE VERSIONAMIENTO





**STABLE TRUNK / STABLE
BRANCH**

STABLE TRUNK (TRONCO ESTABLE) / STABLE BRANCH (RAMA ESTABLE)

Lo que se entiende por rama o tronco estable, es que ha salido una bifurcación o subnivel, otra rama de una de ellas, y es donde se realizan las correcciones, errores, actualizaciones.

El hacer una rama o tronco estable, significa que esa ruta no será afectada, hasta que se validen los cambios por testers y se pueda realizar un merch.



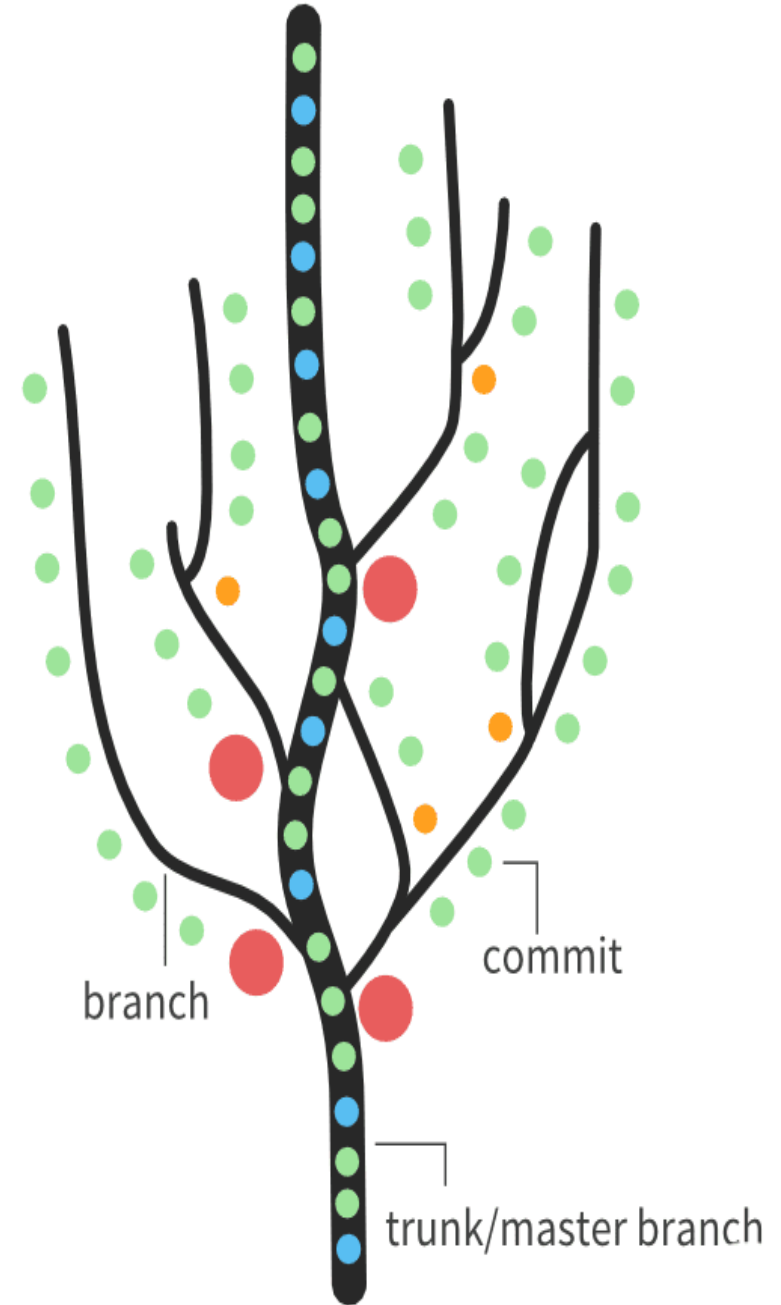


**UNSTABLE TRUNK /
UNSTABLE BRANCH**

Unstable Trunk = Tronco Inestable

Stable Branch = Rama Estable

Esto hace referencia a las distintas ramificaciones en un proyecto, en este caso, al comienzo inestable del proyecto (o una sección importante, trunk/tronco), y como mediante las ramificaciones de este (branches/ramas) solucionar los posibles errores o inconsistencias.





AGILE

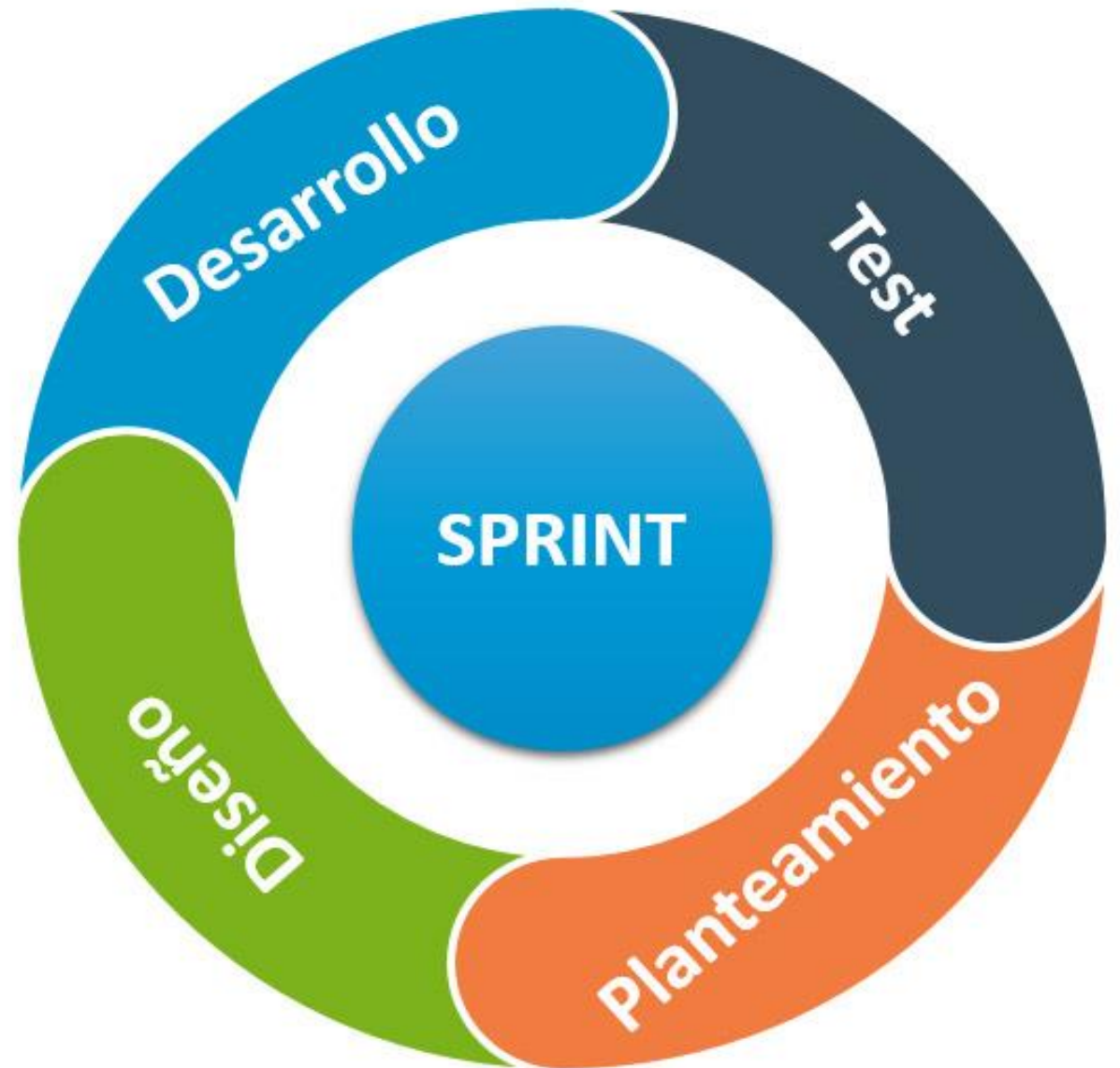
¿QUÉ ES AGILE?

Ágil (del inglés Agile), hace referencia al desarrollo ágil.

El desarrollo de software ágil es un concepto usado en el desarrollo de software para describir las metodologías de desarrollo incrementales.

Es una alternativa en la gestión tradicional de proyectos TI.

Se hace hincapié en el empoderamiento de las personas para colaborar y tomar decisiones en equipo.



TRADICIONAL



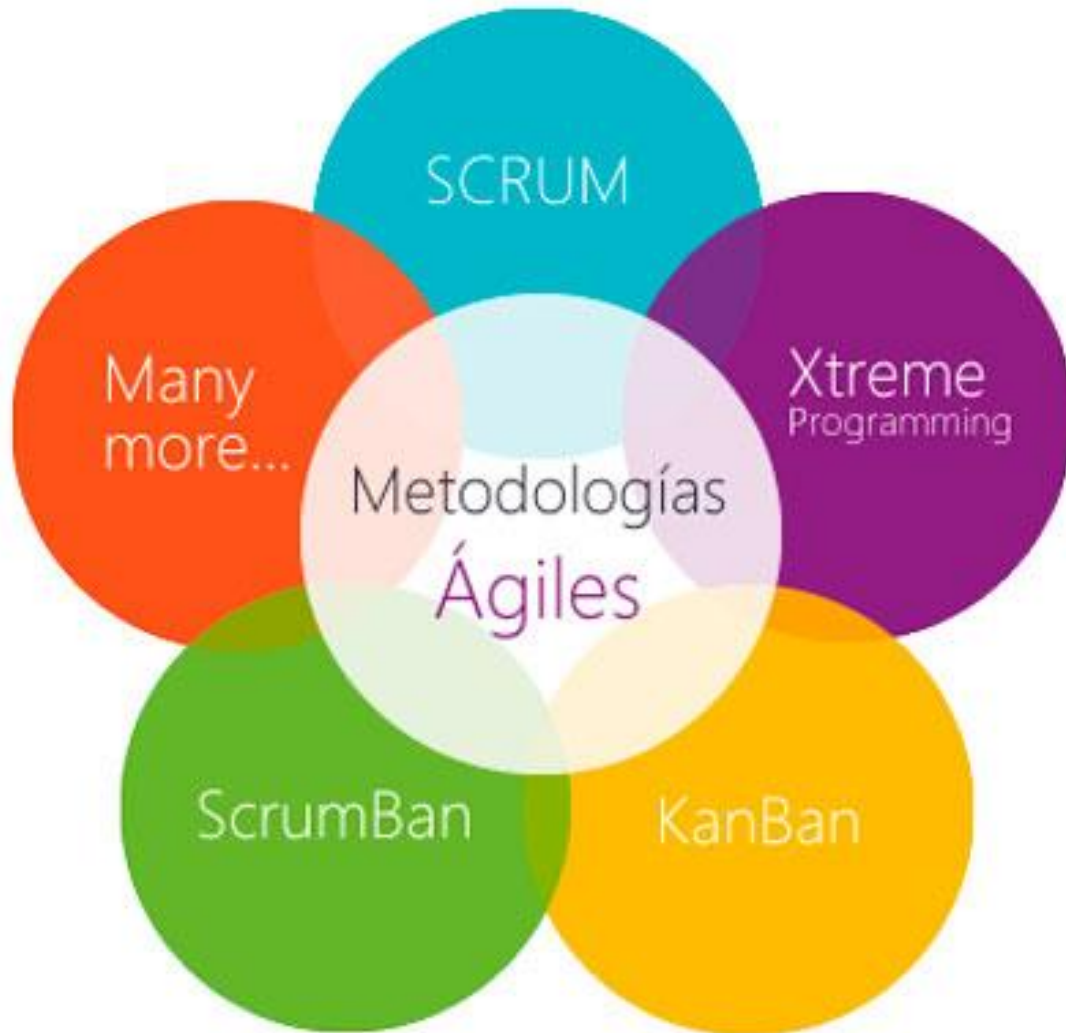
AGILE



RECOMENDACIONES DE UNA METODOLOGÍA DE SOFTWARE ÁGIL

1. Es imperativa la participación activa de los usuarios.
2. El equipo de desarrollo debe tener la facultad para tomar decisiones.
3. Los requisitos evolucionan, pero la escala de tiempo y fechas de entregas son fijas (control del alcance).
4. Capturar los requisitos a un alto nivel, ligero y visual (prototipos).
5. Desarrollar versiones pequeñas, incrementales e itere sobre ellas.
6. Enfocarse en la entrega frecuente de productos.
7. Completar cada funcionalidad antes de pasar a la siguiente.
8. Aplicar la regla 80/20, trabajar funcionalidades principales – principio de Pareto
9. Las pruebas se integran en todo el ciclo de vida del proyecto – prueba temprano y con frecuencia.
10. Un enfoque de colaboración y cooperación entre todas las partes interesadas, es esencial.

METODOLOGÍAS ÁGILES



Actualmente se conocen diferentes metodologías relacionadas con el agilismo, como una respuesta a la creciente necesidad de la industria por entregar productos de calidad en el menor tiempo y costo posible.

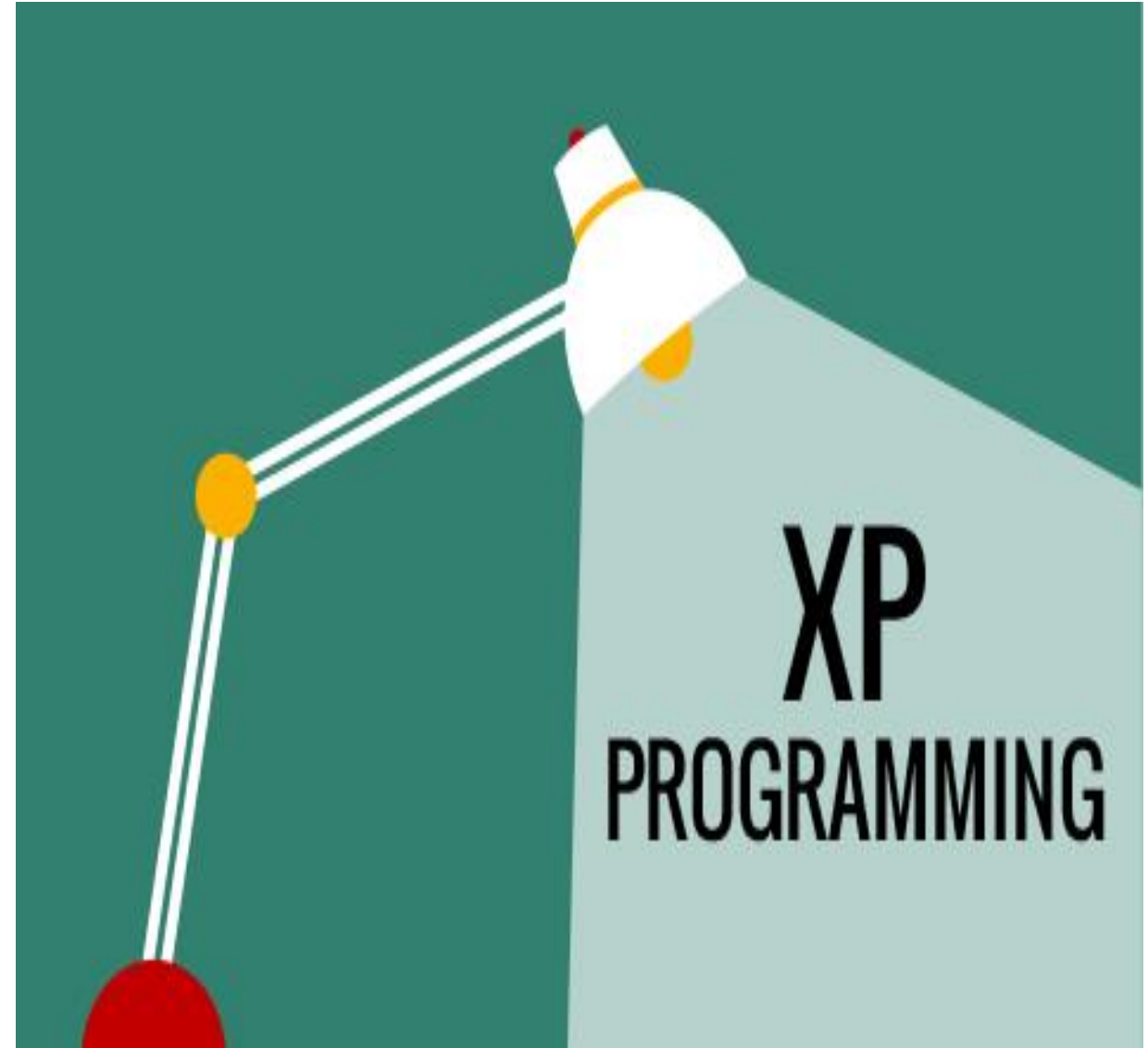
Entre ellas, se pueden mencionar:

Extreme programming (XP), Scrum, Lean, Kanban, Feature Driven Development (FDD), Agile Unified Process (AUP), Agile Modeling, Crystal Clear Methods, Dynamic Systems Development Methods (DSDM).

EXTREME PROGRAMMING (XP)

Metodología que se centra en el desarrollo, en lugar de aspectos de gestión de proyectos. XP fue diseñada para que las organizaciones tuvieran libertad de adoptar la totalidad o parte de la metodología.

Cuando el producto tiene suficientes características para satisfacer a los usuarios, el equipo termina la iteración y libera el software.



SCRUM

Fue propuesta por Ken Schwaber y Jeff Sutherland en 1995. Actualmente es la metodología ágil más usada y extendida en el mundo.

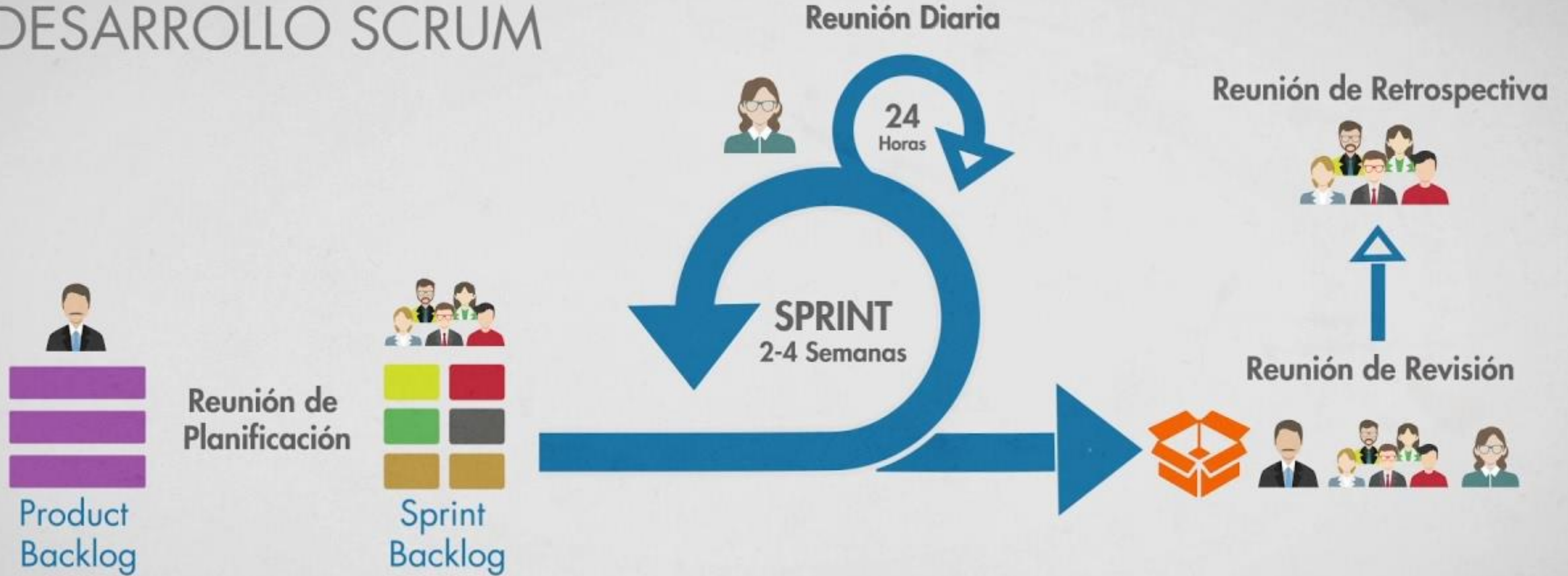
Surge de las iniciativas de prototipado rápido, bajo un entorno en que los requisitos se encuentran incompletos en el inicio y son cambiantes durante el desarrollo. A diferencia de XP, la metodología Scrum incluye tanto los procesos de gestión como de desarrollo.



Scrum.org

Improving the Profession of Software Development

DESARROLLO SCRUM



KANBAN

La metodología se enfoca en mejorar la visibilidad del flujo de trabajo, limitar el trabajo en curso de acuerdo con la capacidad disponible y medir el tiempo de ciclo de vida de una actividad.

Para ello se divide el trabajo en bloques, cada actividad se clasifica dentro de alguno de los bloques y se ubica en un tablero general, el cual se conoce como tablero Kanban.







VERSIONAMIENTO Y TESTING

TRADICIONAL VS AGILE

Desarrollo Tradicional

En los proyectos software tradicionales era muy común el ciclo de vida en cascada o waterfall. Replicando los procesos del mundo de la ingeniería civil, la ingeniería del software abrazó sus dogmas. Análisis, Diseño, Programación, Pruebas, Implantación y periodos de Post-implantación y Garantía.

Desarrollo Agile

En el desarrollo ágil, generalmente utilizamos un enfoque iterativo e incremental. Podemos ir construyendo el producto y evolucionarlo con el feedback que recibimos de sus usuarios.

¿QUÉ ES EL AGILE TESTING?

Cuando se habla de Agile Testing, hay una serie de características que van implícitas en el concepto:

- Es una actividad colaborativa que ocurre continuamente, desde el nacimiento de un producto hasta la entrega y más allá.
- Es un factor clave para la entrega frecuente de valor a nuestros clientes.
- Está enfocado en construir un producto de calidad, utilizando bucles de feedback cortos para validar nuestras hipótesis (aprendizaje validado).
- Guía el desarrollo con ejemplos concretos.
- Las prácticas refuerzan la idea de que la calidad es responsabilidad de todo el equipo.
- Como ves, se requiere de un cambio de mindset, el cual te explicaré en el siguiente punto.

EL “AGILE TESTING” MANIFESTO

Samantha Laing y Karen Greaves crearon en 2013 el siguiente manifiesto sobre el testing.

Valoramos:

- Testing durante todo el proceso **sobre** testing al final.
- Prevenir bugs **sobre** encontrar bugs.
- El entendimiento funcional **sobre** chequear funcionalidad.
- Construir el mejor sistema **sobre** romper el sistema.
- La responsabilidad del equipo por la calidad **sobre** la responsabilidad del tester.

THE TESTING *Manifesto*



we value:

Testing
throughout
OVER
testing at
the end


Preventing
bugs
OVER
finding
bugs

Testing
understanding
OVER
checking
functionality

Building the
best system
OVER
breaking the
system

Team
responsibility
for quality
OVER
tester
responsibility

VIDEOS COMPLEMENTARIOS

A cartoon illustration of a man with brown hair, wearing a dark suit, white shirt, and red tie. He is pointing his right index finger upwards towards the text.

Qué es un

**control de
versiones**





git

EN 2 MINUTOS

CONCLUSIONES

Raúl Alejandro De los Santos Anguiano.

En todo tipo de software es muy importante lo que se hace en ellos y lo que es el software pero algo que es siempre muy importante es el versionamiento de estos o donde nosotros podemos saber que podríamos dejar el software muchas de las ocasiones muchos programadores siempre tienen todo de forma remota y si eso está bien ya que si lo hace más seguro pero viéndolo de forma un poco mas laboral cabe decir que pasa si le pasa algo a la computadora o que se rompa o se le meta un virus y que en casos muy drástico el reiniciarla lo cual haría que todo el trabajo de meses quede en el adiós ya que con se perderían horas y horas o años de trabajo por eso la importancia de subir todo a los repositorios ya que estos nos ayudan a tener de una forma segura y que si pasa algo con nuestras computadoras, nuestros trabajos estarán a salvo donde cada uno puede estar bien.

José Felipe Muñiz Fonseca.

El uso de un controlador de versiones hoy en día es fundamental, ya que todo proyecto de software es desarrollado por múltiples personas, lo cual facilita la programación de este y la unificación del mismo.

Es importante saber manejar correctamente las aplicaciones que nos permiten el versionamiento del proyecto, ya que, además de ayudar al desarrollo de una aplicación también nos brindan herramientas importantes a la hora de hacer pruebas, encontrar bugs y solucionarlos; todo con el objetivo de mantener la funcionalidad y calidad de este.

Para concluir, un controlador de versiones así como las distintas metodologías de desarrollo son fundamentales a la hora de crear un proyecto y testarlo, para así mantener la calidad de este durante todo el proceso y garantizar la calidad al usuario.

Néstor Josué Puentes Inchaurregui.

El control de versiones, es una genialidad de herramienta, porque no solo te detalla los cambios que se hicieron, sino que puedes tener la seguridad de que en algún momento puedes regresar a ese cambio, su sistema de crear ramas para trabajar sin comprometer el tronco o rama principal, es excepcional porque tu realizas cambios de manera normal, sin preocuparse a corromper todo el proyecto, y además que facilita que más de un desarrollador este trabajando en su parte en su propia rama y también con solo una línea de comando poder actualizar los cambios que otros desarrolladores han hecho (*claro de los cambios efectuados en la rama sobre la que trabajas*).