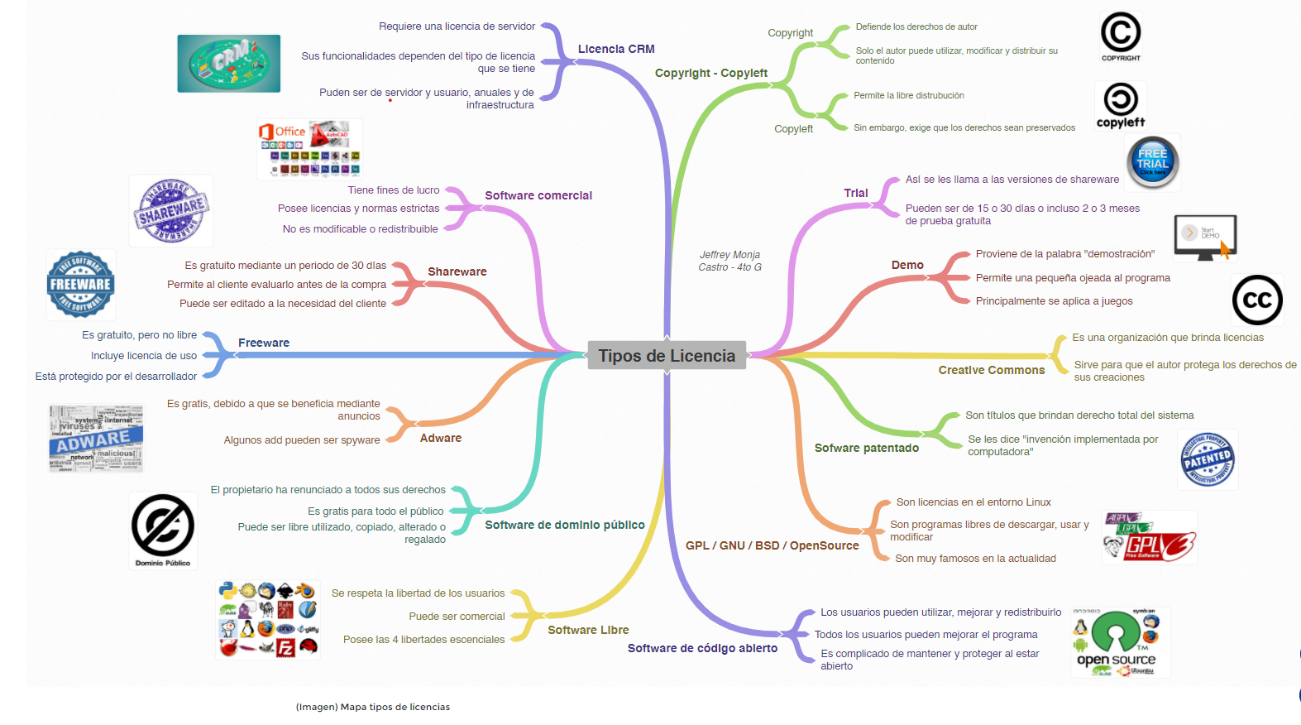
**ARGENTINA PROGRAMA**

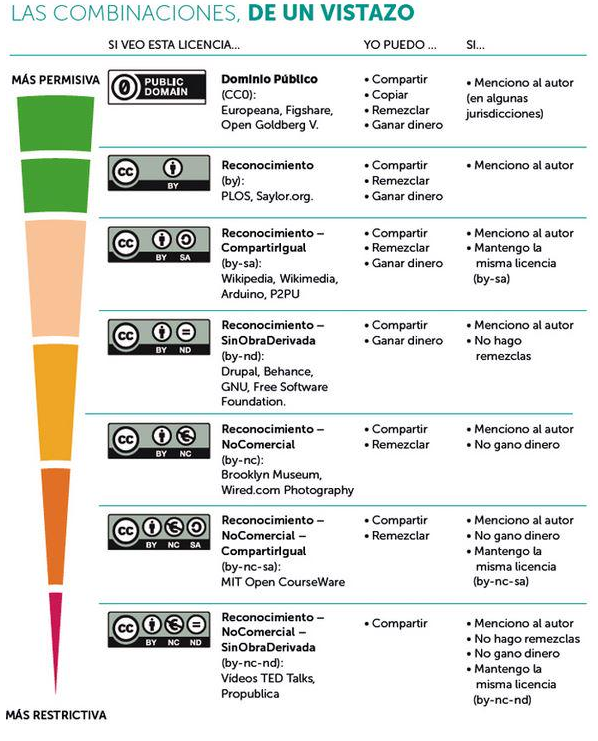
*Herramientas para programadores:*

Empecemos por algunos puntos que nos van a ser útiles…

Traductor:

El clásico es GoogleTranslate. Otra opción recomendable es DeepL.





**Stack Overflow** es una comunidad abierta para cualquiera que codifique. Lo ayudamos a obtener respuestas a sus preguntas de codificación más difíciles

Glosario de recursos útiles

**Cursos de Programación (y más):**

Udacity: <https://www.udacity.com/>

Coursera: <https://es.coursera.org/>

Canal Youtube MIT: <https://www.youtube.com/c/mitocw>

EdX: <https://www.edx.org/es/>

**Herramientas CSS:**

CSS Gradient: <https://cssgradient.io/>

CSS Layout: <https://csslayout.io/>

Getwaves: <https://getwaves.io/>

WebGradients: <https://webgradients.com/>

Cubic-bezier: <https://cubic-bezier.com/#.17,.67,.83,.67>

CSS3: <https://www.css3.me/>

CSS-Minifier: <https://cssminifier.com/>

**Herramientas BootStrap:**

Icons.getbootstrap: <https://icons.getbootstrap.com/>

Themes.getbootstrap: <https://themes.getbootstrap.com/>

Bootswatch: <https://bootswatch.com/>

Getbootstrap: <https://getbootstrap.com/>

**Herramientas HTML:**

HTML-minifier: <https://html-minifier.com/>

HTML ColorCode: <https://htmlcolorcodes.com/es/>

W3schools: <https://www.w3schools.com/>

**Herramientas e Imágenes Alta resolución:**

Freepik: <https://www.freepik.es/>

Pexels: <https://www.pexels.com/es-es/>

MotionArray: <https://motionarray.com/>

UnDraw: <https://undraw.co/>

UnSplash: <https://unsplash.com/>

FreeImages: <https://www.freeimages.com/es>

Pixabay: <https://pixabay.com/es/>

Thenounproject: <https://thenounproject.com/>

Tinypng: <https://tinypng.com/>

Manypixels: <https://www.manypixels.co/>

Humaans: <https://www.humaaans.com/>

Uigradients: <https://uigradients.com/#CocoaaIce>

Flaticon: <https://www.flaticon.es/>

Boxicons: <https://boxicons.com/>

**Herramientas FrontEnd:**

Codepen: <https://codepen.io/>

Modernizr: <https://modernizr.com/>

BrowserShots: <https://browsershots.org/>

CDNJS: <https://cdnjs.com/>

Waybackmachine: <https://archive.org/web/>

Dummyimage: <https://dummyimage.com/>

Colorzilla: <https://www.colorzilla.com/>

Caniuse: <https://caniuse.com/>

Spritecow: <http://www.spritecow.com/>

Figma: <https://www.figma.com/>

JavasScript-minifier: <https://www.minifier.org/>

**Fuentes:**

fonts.google: <https://fonts.google.com/>

Es.lipsum: <https://es.lipsum.com/>

FontAwesome: <https://fontawesome.com/>

**Editores de código:**

VScode: <https://code.visualstudio.com/>

**VCS:**

Subversión: <https://subversion.apache.org/>

Mercurial: <https://www.mercurial-scm.org/>

Git: <https://git-scm.com/>

Github: <https://github.com/>

Gitlab: <https://about.gitlab.com/>

Bitbucket: <https://bitbucket.org/>

**Programas para VCS:**

Github-desktop: <https://desktop.github.com/>

Git-kraken: <https://www.gitkraken.com/>

**Sistema de persistencia:**

SQLite: <https://www.sqlite.org/index.html>

MariaDb: <https://mariadb.org/>

MongoDb: <https://www.mongodb.com/es>

Oracle: <https://www.oracle.com/ar/index.html>

PostgreSQL: <https://www.postgresql.org/>

Redis: <https://redis.io/>

MySQL: <https://www.mysql.com/>

DynamoDb: <https://aws.amazon.com/es/dynamodb/>

AmazonRelationDataBaseService: <https://aws.amazon.com/es/rds/>

**Navegadores:**

Chrome: <https://www.google.com/intl/es-419/chrome/>

Brave: <https://brave.com/es/>

Chrome Canary: <https://www.google.com/intl/es-419/chrome/canary/>

Chromium: <https://www.chromium.org/>

Firefox Developer Edition: <https://www.mozilla.org/es-AR/firefox/developer/>

**Servicios web:**

Rest API

Graphql API – Serverless Firebase

AWS Amplify

Cloudinary

**Desarrollo API:**

Postman: <https://www.postman.com/product/graphql-client/>

Rest Client de VSCode: <https://marketplace.visualstudio.com/items?itemName=humao.rest-client>

**Servicios cloud:**

Azure: <https://azure.microsoft.com/es-es/>

Amazon Web Services: <https://aws.amazon.com/es/>

GoogleCloud: <https://cloud.google.com/>

Docker - para empaquetar: <https://www.docker.com/>

**CMS:**

Wordpress: <https://wordpress.com/es/>

Woocommerce: <https://woocommerce.com/>

Magento: <https://magento.com/>

Blogger: <https://www.blogger.com/about/>

Ghost: <https://ghost.org/>

**Framework de Backend:**

GO (Gorilla, Buffalo, goji)

Python (django, flask)

Typescript/JavaScript/Node (Loopback, Nest, Next.js, Nuxts.js)

**Servers:**

Nginx: <https://www.nginx.com/>

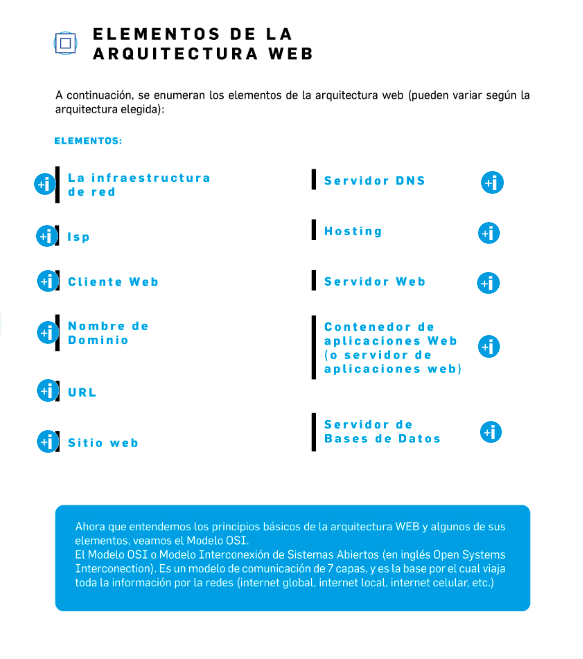
Apache: <https://www.apache.org/>

Windowsiis: <https://www.iis.net/>

Tomcat: <http://tomcat.apache.org/>

**Mobiledev:**ionic (angular react vim )-react native + expo, flutter

**ARQUITECTURAS WEB**

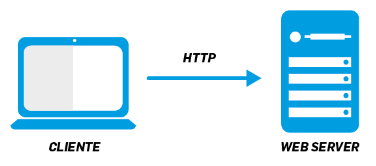
A partir del desarrollo de ARPANET en 1969 empieza un crecimiento vertiginoso del uso de la internet. En 1990 Tim Berners-Lee creó la WWW , la "WorldWideWeb" que realizó la primera conexión desde un navegador a un servidor web mientras trabajaba en el CERN desarrollando así, las tres tecnologías fundamentales de la web que son:

● HTML (lenguaje de marcado de hipertexto). Lenguaje de marcado o etiquetado que se emplea para escribir los documentos o páginas web.

● URL (localizador de recursos universal). El localizador de recursos uniforme, sistema de localización o direccionamiento de los documentos web.

● HTTP (Protocolo de transferencia de hipertexto) El lenguaje con el que se comunica el navegador con el servidor web y el que se emplea para transmitir los documentos web.

Se trata entonces de una arquitectura cliente-servidor en la que cada dispositivo electrónico en la red ( internet , intranet o extranet ) actúa como cliente o servidor lo que implica la comunicación entre procesos que hacen peticiones (clientes) y procesos que responden a esas peticiones (servidores). Esta comunicación es posible gracias al protocolo HTTP.



**Arquitectura cliente / servidor básico**

En 1994 (1 de octubre) Tim Berners-Lee abandona el CERN y funda la W3C , en inglés, "World Wide Web Consortium", organismo internacional que propone recomendaciones y estándares web que aseguran el crecimiento de la World Wide Web.

Haciendo clic aquí podrás ver la evolución de la web. Mencionaremos los hitos más relevantes:

● En 1991 surge HTTP definido como "protocolo de red para sistemas de información hipermedia distribuidos".

● Muy próximo aparece HTML 1, es el lenguaje de marcado predominante de las páginas web.

● En 1995, Netscape creó JavaScript , un lenguaje de secuencias de comandos basado en prototipos y "orientado a objetos". El objetivo de este lenguaje de programación fue darle capacidad de ejecución al cliente de esta arquitectura web, es decir, al navegador.

● En 1998 aparecen las hojas de estilo, en su versión 2. Se denominaron CSS , del inglés "Cascading Style Sheets", que es un lenguaje de hojas de estilo empleado para describir la semántica de presentación de un documento, en este caso un documento web

**Arquitectura de las aplicaciones Web**

Las aplicaciones web se basan en una arquitectura cliente / servidor. Es decir que, por un lado está el cliente (navegador) y por otro lado el servidor. Existen diferentes variantes de la arquitectura básica según como se implementa, pero es importante mencionar que en la tecnología la mayoría de las estructuras están compuestas por capas.

Algunas de las arquitecturas más comunes:

***Servidor web + base de datos en un mismo servidor*** (2 niveles o capas). En este caso el servidor gestiona tanto la lógica de negocio como la lógica de los datos y los datos.

***Servidor web y de datos separados*** (3 niveles). En este caso se separa la lógica de negocio a la de datos en diferentes servidores.

***Servidor web + servidor de aplicaciones + base de datos*** en un mismo servidor (4 niveles).

Cada capa puede estar en un servidor diferente y aun así se pueden interconectar.

Un grupo de páginas web dinámicas se conceptualiza como Front End (pensadas para que el cliente acceda) y el otro grupo de páginas dinámicas web como Back End (pensadas para el procesamiento y acceso a datos), además de que la base de datos puede existir en otro Servidor. Esto da lugar a un concepto muy importante en POO el DESACOPLAMIENTO, en este caso el diseño del Front End, Back End y Base de Datos puede desacoplarse.

En el escenario 4: Vemos una Plataforma Digital o un Ecosistema, en este caso podemos ver que consiste en muchos sistemas que trabajan en conjunto, colaborando para resolver una necesidad o problema. En el ejemplo de Mercado Libre se dedica a vender online, pero también tiene que cobrar y hacer envíos. Por eso han desarrollado aplicaciones independientes pero que saben cómo comunicarse a otros sistemas para pedir o enviar datos para realizar alguna tarea. De esta manera las aplicaciones pueden dar solución integral, comprar, pagar y enviar el producto sin tener que salir de la página. Esta arquitectura es más compleja porque son varios los sistemas que componen el ecosistema, pero si miras con atención el grafico, notaras que existen los mismos elementos que describimos anterior, revisemos:

**Front End:** Parte de la aplicación web que ve un usuario al entrar.

**Back End:** Parte de la aplicación web que no se ve y que realiza las acciones en el servidor.

**Conexión a BD:** En este caso se representa con un icono de una base de datos.

**Apis:** Parte de la aplicación que permite conectarse a otras aplicaciones y que otras aplicaciones se conecten con nuestro sistema.

**Arquitectura Web distribuida**

**¿Qué es una arquitectura centralizada?**

Es cuando tenemos todos los elementos de nuestra aplicación web de arquitectura de Cliente / Servidor en un solo lugar equipo o servidor, es decir, tener el Back End, Front End, Bases de datos y APIs en el mismo equipo. Esto hace que en el caso de una falla del equipo toda nuestra aplicación también fallará.

**¿Qué es la arquitectura distribuida?**

Es tener la posibilidad y capacidad de separar nuestro sistema en distintos servidores de la red (sea red local o internet).

**¿Pero cómo te das cuenta cuando una aplicación es distribuida?**

Como usuario: No te darías cuenta porque si el sistema está distribuido funciona como un conjunto único y sincronizado.

Como Programador: Cuando te asignen a un proyecto o cliente en base a preguntas concretas podrás ir conociendo cómo se implementó o distribuyo el sistema, pero te compartimos algunas formas en las que puedes entender que estás frente a una arquitectura distribuida:

Por la Líder del Proyecto: Cuando se comienza a trabajar en un proyecto generalmente la líder del mismo hace una explicación del tipo de aplicación con la que se está trabajando, además de indicarnos en qué parte del proyecto estaremos trabajando.

Por el perfil asignado: Cuando nos asignan el trabajo en una empresa nos especifican si trabajaremos en el Front End, en el Back End o en ambos Full Stack, de esa manera podemos inferir que la arquitectura es distribuida, igualmente siempre es mejor preguntar para estar seguros.

Por un diagrama: Generalmente se utilizan diagramas de aplicación para documentar un sistema, en el se puede ver la separación del sistema y si esta distribuido en 1 o varios servidores.

**¿Hasta dónde puedo modularizar o distribuir mi sistema?**

El cómo distribuir el sistema es algo que se analiza en el diseño de la aplicación o se va cambiando a medida que el sistema va creciendo.

Cuando una aplicación se hace más grande, compleja y con más usuarios necesitamos seguir modularizando el sistema, dado que no nos alcanza con separar en Front End, Back End y Base de datos. En esta situación ya debemos pensar en modularizar o separar algunas funcionalidades del sistema, algunos motivos pueden ser:

Por alta demanda: Cuando el sistema tiene una funcionalidad que es compleja, consume mucho recurso del servidor o es muy demandada por distintas partes del sistema.

Por interconexión: Cuando un sistema tiene funcionalidades que se necesita dar acceso a otros sistemas para consumir ese proceso, función o datos.

Por segregación de roles: Cuando es necesario separar roles o funciones por motivos de seguridad o aspectos técnicos, también puede ser porque negocio lo requiere, por ejemplo si se decide por seguridad separar el proceso de autenticación del sistema para reforzar la seguridad.

Por escalamiento: Cuando las proyecciones indican que en un periodo de tiempo la demanda aumentará considerablemente, será necesario agregar más servidores en la red con la misma funcionalidad para que satisfaga la demanda.

Existen varias formas de separar estas funcionalidades que llamaremos API REST o Microservicios.

Para ejemplificar tomaremos un sistema que tiene lo siguiente:

***Front End:*** Todos fue diseñado pensando en una arquitectura distribuida, las funcionalidades son las siguientes.

Login: Se conecta con el Back End para validar el usuario y clave.

Cliente: Se conecta con el Back End para consultar, editar, crear y eliminar clientes

Producto: Se conecta con el Back End para consultar, editar, crear y eliminar productos

Proveedor: Se conecta con el Back End para consultar, editar, crear y eliminar proveedores

***Back End:*** Todos fue diseñado pensando en una arquitectura distribuida, las funcionalidades son las siguientes,

Login: Recibe la petición, consulta la base de datos y valida si el usuario existe.

Cliente: Recibe la petición, consulta la base de datos y devuelve los datos de cliente.

Producto: Recibe la petición, consulta la base de datos y devuelve los datos del producto.

Proveedor: Recibe la petición, consulta la base de datos y devuelve los datos del proveedor.

**Elementos de la arquitectura Web**

**1- La infraestructura de red**

Si bien es cierto que, en fase de desarrollo, para probar nuestra aplicación web, no necesitaríamos esta infraestructura, una vez nuestra aplicación se instale en el hosting definitivo, será necesario una red ethernet y todos los componentes que hacen posible la conectividad de los equipos informáticos. Con esto nos referimos a cables de red, sea utp o fibra óptica, placas de red, sea wifi o cableada, switch, routers, modem, etc. Y estos componentes físicos son necesarios tanto del lado del cliente como del servidor. Con esto se quiere decir que no es posible implementar una aplicación web sino existe una infraestructura de red preexistente o se diseña e implementa una nueva. Esta puede ser: internet, intranet o extranet.

**2.- Isp**

Es el proveedor del servicio de internet.

**3.- Cliente Web**

Es el navegador web. Ejemplos: Chrome, Safari, Firefox, etc. Pero ya no se restringe solo a estos dispositivos sino que podría ser, por ejemplo, un sistema embebido ejecutándose en una SBC (small board computer). Incluso podría no estar ejecutando un navegador convencional, como por ejemplo un reloj inteligente, o un dispositivo vinculado a una máquina de producción seriada como los surgidos de la mano del concepto de Industria 4.0.

**4.- Nombre de Dominio**

Dicho de forma sencilla, el nombre de dominio (o simplemente "dominio") es el nombre de un sitio web. Es decir, es lo que aparece después de "@" en una dirección de correo electrónico o después de "www." en una dirección web. Si alguien te pregunta cómo te puede encontrar en Internet, tendrás que decirle tu nombre de dominio. Las computadoras para comunicarse utilizan direcciones IP (números únicos en la red). Un ejemplo de una dirección IP de un servidor es 173.194.121.32. Para nosotros es imposible recordar tantos números y saber qué servicio o qué aplicación se encuentrta en esa dirección IP o servidor. Para resolver estos problemas se usan palabras que las personas pueden leer, que son intuitivas, fáciles de recordar y dicen mucho sobre el servicio web que ofrecen, se denominan nombres de dominio. ¿Puedo comprar un nombre de dominio? No, los nombres no se pueden comprar, solo se puede pagar por el derecho a usarlo por cierto periodo de tiempo. Para registrar un dominio a tu nombre debes hacerlo por medio de una empresa que se encarga de administrar las registraciones de nombres de dominio. En el caso de Argentina es https://nic.ar

Aquí tienes algunos ejemplos de nombres de dominio: google.com, wikipedia.org, youtube.com https://domains.google/intl/es\_es/learn/web-terms-101/

**5.- URL**

Una URL (o localizador uniforme de recursos) es una dirección web completa que se utiliza para encontrar una página web específica. Mientras que el dominio es el nombre del sitio web, la URL es una dirección que remite a los usuarios a una de las páginas de ese sitio web. Cada URL contiene un nombre de dominio y otros componentes necesarios para localizar una página o un contenido concreto. Aquí tienes algunos ejemplos de URL: http://www.google.com - https://es.wikipedia.org/wiki/umami - https://www.youtube.com/feed/trending ; https: // dominios. google / intl / es\_es / learn / web-terms-101 /

**6.- Sitio web**

Aunque una cosa lleve a la otra, comprar un nombre de dominio no implica tener un sitio web. El dominio es el nombre del sitio web, la URL es la forma de encontrarlo y el sitio web es lo que los usuarios ven en su pantalla y con lo que interactúan. Es decir, cuando compres un dominio, habrás adquirido el nombre de tu sitio web, pero te faltará crear el sitio web en cuestión. https://domains.google/intl/es\_es/learn/web-terms-101/

**7.- Servidor DNS**

Sistema de Nombre de Dominio. Se ocupa de la administración del espacio de nombres de dominio. Este servidor se encarga de hacer las conversiones de nombres de dominio a direcciones IP. Cuando el cliente realiza una petición web, por ejemplo google.com, una de las primeras acciones del sistema es invocar un servidor DNS para que devuelva la dirección IP del / o de alguno de los servidores de google. Por ejemplo devolverá la ip 172.217.162.14.

**8.- Hosting**

Es el nombre que se le da al servicios de alojamiento en la web a nuestras paginas, aplicaciones, bases de datos (los hosting son servidores que están siempre encendidos y conectados a internet). Los programadores una vez terminado el trabajo suben su aplicación web al Hosting para que todo el mundo pueda acceder.

**9.- Servidor Web**

También llamado servidor HTTP, es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente.

**10.- Contenedor de aplicaciones Web (o servidor de aplicaciones web)**

Es el módulo que permite la ejecución de aplicaciones web. Por ejemplo el módulo PHP o Python del Servidor Web. Componente ASP o ASPX de IIS. Servidor o Contenedor de Aplicaciones Web Java: Tomcat, Weblogic, Websphere, JBoss, Geronimo, etc.

**11.- Servidor de Bases de Datos**

Estos son contenedores de bases de datos que permiten organizar y administrar los datos que deben permanecer en un medio de almacenamiento permanente. Resuelven problemas de: seguridad, mecanismos de comunicación, concurrencia, inconsistencias de los datos, respaldo, entre otros. Hay varios tipos de bases de datos, por ejemplo, las relaciones que organizan los datos en forma de tablas, en filas y columnas. Otro tipo son los orientados a objetos u orientados a documentos donde el concepto de tablas se cambia por la colección con formatos similares a “json”. JavaScript Object Notation (JSON) es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript. Es utilizado para transmitir datos en aplicaciones web (por ejemplo: enviar algunos datos desde el servidor al cliente, así estos datos pueden ser mostrados en páginas web, o viceversa)

**Full Stack – Infografía**

El desarrollador web full stack puede crear aplicaciones web dinámicas. Esto se logra en base a los estándares web y utilizando tecnologías web que pueden variar según la pila de desarrollo. Tiene por objetivo la creación de aplicaciones web dinámicas.

Pero ¿qué son los estándares web y qué buscadores existen? Los estándares web son tecnologías que se utilizan para crear aplicaciones web. Los mismos son creados por organismos de estándares - instituciones que invitan a grupos de personas de diferentes compañías de tecnología a unirse y acordar cómo deben funcionar las tecnologías de la mejor manera posible para cumplir con todos sus casos de uso. El W3C es el organismo de estándares web más conocido, pero hay otros como WHATWG (responsables de la modernización del lenguaje HTML), ECMA (publica el estándar para ECMAScript, en el que se basa JavaScript), Khronos (publica tecnologías para gráficos 3D, como WebGL) y otras MDN Web Docs

**¿ Pero qué son los Stacks?**

Lo que denominamos stack tecnológico, o también denominado stack de soluciones o ecosistema de datos, es un conjunto de todas las herramientas tecnológicas utilizadas para construir y ejecutar una sola aplicación.

Recordemos lo que ya hemos visto en la arquitectura Cliente /Servidor a su vez al código que navega el cliente por el navegador le llamamos Front End y la parte de código que accede a los datos se le llama Back End

Cada una de estas partes de la arquitectura puede crearse con distintos lenguajes de programación, el que quieras, por citar algunos ejemplo:

Front End: Para el desarrollo del Front End se necesita.

Estructura y estilos: HTML, CSS o framework como Bootstrap

Lenguaje programación: JavaScript, java, PHP o bien pueden ser framework como Angular

Back End: Para el desarrollo de Back End se necesita un web server, una base de datos y un lenguaje de programación.

Base de datos: puede ser MySQL, PostgreSQL, etc

Lenguaje: Java, PHP, etc

Web server: Apache, NGINX, etc.

**¿Qué significa ser un desarrollador full stack?**

El desarrollador full stack es un perfil que tiene conocimiento de lenguajes de programación de Front End, Back End, APIs y Bases de datos. Esta amplitud de conocimiento es muy buscada hoy en día por las empresas, esto significa que al momento de trabajar en un proyecto podrías estar asignado a cualquiera de esas áreas.

**Proceso de una petición web**

1. Cliente Web: Solicita la resolución de nombres al servidor DNS. Por ejemplo: google.com
2. Servidor DNS: Recibe y trata la solicitud. Una vez recibida la petición realiza las consultas necesarias para resolver y obtener la dirección IP.
3. Servidor DNS: Devuelve al navegador Web la dirección IP que corresponde al Servidor Web.
4. Cliente Web: Conecta con el servidor web mediante la dirección IP y el puerto. Realiza la petición mediante una URL (Método GET) o un formulario (Método POST). Dicha solicitud incluye: la dirección IP del servidor web, el puerto del servidor web, URL y parámetros.
5. Servidor Web: Control de Acceso, Análisis de la petición y localización del recurso. Como detecta que es el acceso a un fichero o ruta de aplicación tiene que traspasar el control al Contenedor de aplicaciones Web
6. Paso de la petición del servidor web al contenedor de aplicaciones web
7. El contenedor analiza la petición y en base a la ruta traspasa el control a la aplicación web.
8. Paso del control de la petición desde el CAW a la aplicación.
9. La aplicación recibe la petición y decide qué hacer en base a ella, es decir, elegir la función que se encargará de gestionar esa petición, normalmente en base a la ruta, el método HTTP y los parámetros de entrada por URL. Una vez elegida ejecutará esa función.
10. La aplicación realiza una petición SQL a la base de datos.
11. La Base de Datos recibe la petición SQL y la procesa realizando los cambios que tenga que hacer, si corresponde.
12. Una vez procesada la petición devuelve los datos a la aplicación web, normalmente un conjunto de datos. Ej. los 10 últimos clientes.
13. La aplicación web recibe estos datos y tiene que generar una salida, normalmente HTML, donde estructura el contenido de los datos devueltos por la BBDD en etiquetas HTML.
14. La aplicación web devuelve una respuesta al Contenedor de Aplicaciones Web
15. El contenedor procesa la respuesta, para controlar la ejecución de la aplicación por si esta falla.
16. El Contenedor de Aplicaciones Web devuelve el fichero al servidor web.
17. El servidor Web devuelve los datos dentro de la respuesta HTTP al navegador web.
18. Cliente Web: Presenta (renderiza) el contenido HTML resultante.

**Transferencia de datos**

**Conceptualización de las estructuras involucradas en la transferencia de datos entre computadoras**Para comenzar vamos a definir que “ (...) una red de computadoras, también llamada red de ordenadores, red de comunicación de datos o red informática, es un conjunto de equipos (nodos) y software conectados entre sí por medio de dispositivos físicos (cable) o inalámbricos (wifi) que envían y reciben impulsos eléctricos, ondas electromagnéticas o cualquier otro medio para el transporte de datos con la finalidad de compartir información recursos y ofrecer servicios (...) ”

**Red del Hogar:** En la red del hogar podemos observar que entra el cable al modem wifi permitiendo que los equipos o nodos se conecten por la red inalámbrica, los dispositivos que se conectan son notebook, cámaras, celulares, impresoras. También podemos ver que desde el modem sale un cable hacia una computadora de escritorio. Todo está conectado por la misma red.

**Red Empresarial:** En la red empresarial entra un cable al modem principal que llega al primer y segundo edificio permitiendo conectar en red a los equipos o nodos (notebook, pc de escritorio, impresora, servidores). También observamos que desde el modem principal sale un cable de red que conecta con un modem wifi el cual permite conectar los equipos o nodos que tenga conexión a la red por medio de wifi.

**Red Empresarial:** En la red de proveedor de internet podemos observar que distintas empresas se conectan entre sí formando una red en la que transmiten y reciben datos.

**¿Cual es el objetivo de una red?**

El objetivo de las redes en general, consisten en "compartir recursos" entre computadoras de modo tal que los programas, datos y equipo estén disponibles para cualquier solicitud que los equipos de la red requieran, sin importar la ubicación o distancia física del recurso (equipo) y del usuario.

La finalidad principal para la creación de una red de ordenadores es:

● Compartir recursos e información a grandes distancias (computadoras, impresoras, servidores, cámaras, etc.)

● Asegurar la confiabilidad y la disponibilidad de la información (características propias que definen lo que es información).

● Aumentar la velocidad de transmisión de los datos y obtenerlos en milisegundos para su uso.

● Reducir los tiempos y costos en actividades como procesamiento de información.

**¿Como podemos entender una red?**

La comunicación por medio de una red se lleva a cabo en dos categorías diferentes: una capa denominada física y otra lógica.

La capa física incluye todos los elementos de los que hace uso un equipo para comunicarse con otro equipo dentro de la red.

La capa lógica la comunicación se rige por normas muy rudimentarias que por sí mismas resultan de escasa utilidad. Sin embargo, haciendo uso de dichas normas es posible construir los protocolos denominados, que son normas de comunicación más complejas (de alto nivel) capaces de proporcionar servicios útiles.

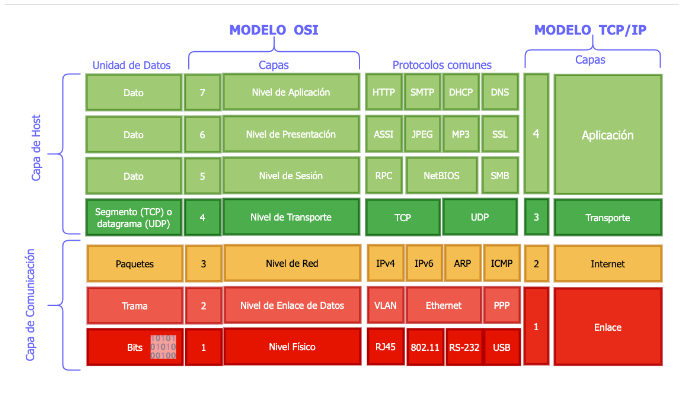
Los protocolos son un concepto muy similar al de los idiomas de las personas. Es decir, si dos personas hablan el mismo idioma, y ​​respetan ciertas reglas (tales como hablar y escucharse por turnos), es posible comunicarse y transmitir ideas e información. Ese es el modelo de un protocolo.

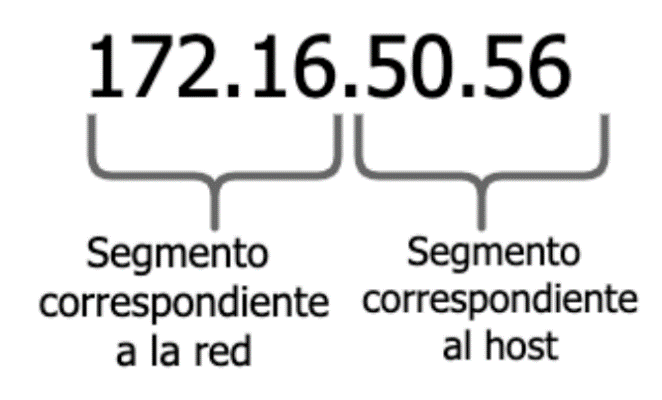
Para formar una red se requieren elementos de hardware, software y protocolos. Los elementos físicos se clasifican en dos grupos: los dispositivos de usuario final (llamados también Hosts) y los dispositivos de red. Entre los dispositivos de usuario final podemos enumerar computadoras, impresoras, escáneres, y demás elementos que brindan servicios directamente al usuario. Los segundos (dispositivos de red) son todos aquellos que conectan entre sí a los dispositivos de usuario finales posibilitando su intercomunicación.

Internet es un conjunto descentralizado de redes de comunicación interconectadas que utilizan la familia de protocolos TCP / IP, lo cual garantiza que las redes físicas heterogéneas que la componen, constituyan una red lógica única de alcance mundial.

Uno de los servicios que más éxito ha tenido en Internet ha sido world wide web, WWW o la web.

Para el envío de correo electrónico usamos el protocolo SMTP, para la transmisión de archivos usamos el protocolo FTP.

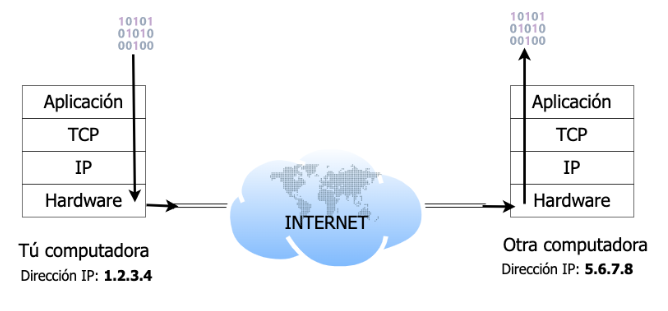
TCP (protocolo de control de transmisión) se usa para crear conexiones entre computadoras a través de las cuales pueden enviarse un flujo de datos. Por la forma en la que está implementado este protocolo los datos serán entregados en su destino sin errores y en el mismo orden en el que se transmitieron.

Por otro lado, el protocolo IP es un protocolo cuya función principal es el uso direccional en origen o destino de comunicación para transmitir datos mediante un protocolo no orientado a conexión que transfiere paquetes conmutados a través de distintas redes previamente enlazadas según la norma OSI.

Entonces, en conclusión, se utiliza la combinación de estos dos protocolos para la comunicación en Internet, en donde TCP aporta la fiabilidad entre la comunicación e IP la comunicación entre distintas computadoras ya que las cabeceras de IP (cabecera por ser una parte el protocolo) contienen las direcciones de destino de las máquinas de origen y llamadas direcciones IP.

Para entender mejor el funcionamiento de la Internet vamos a decir que dentro de la red de redes que es Internet debe existir un mecanismo para conectar dos computadoras. Este mecanismo lo proporciona el protocolo de Internet, el cual hace que un paquete de una computadora, llegue a la otra de manera segura a través del protocolo TCP y que llegue a destino a través de las direcciones IP.

Ya vimos que la estructura de red se maneja en capas. También mencionamos que hay una capa de red en dónde está el protocolo IP, una capa Superior de transporte en dónde está el protocolo TCP y ahora vemos una nueva capa que es la de aplicación en dónde se usa el protocolo HTTP.



**HyperText Transfer Protocol (HTTP)**

Hypertext Transfer Protocol (HTTP) (Protocolo de Transferencia de Hipertexto en español) es un protocolo de la capa de aplicación para la transmisión de documentos hipermedia, como HTML. Fue diseñado para la comunicación entre los navegadores y servidores web, aunque puede ser utilizado para otros propósitos también.

HTTP define un conjunto de métodos de petición para indicar la acción que desea realizar para un recurso determinado. Aunque estos también pueden ser sustantivos, estos métodos de solicitud a veces son llamados verbos HTTP .

Es muy importante saber que HTTP es un protocolo sin estado, es decir, no guarda ninguna información sobre conexiones anteriores. Luego veremos las implicancias de esto.

Una descripción importante del protocolo es que está orientado a transacciones y sigue el esquema de petición / respuesta entre un cliente y un servidor. El cliente realiza una petición enviando un mensaje con cierto formato al servidor. El servidor le envía un mensaje de respuesta. Para hacerlo más concreto, un cliente podría ser un navegador web y un servidor podría ser una aplicación en un servidor web corriendo en Internet.

Para que entendamos la forma de cómo se programa una aplicación web, necesitamos en alguna medida entender los mensajes HTTP. Los mensajes HTTP son en texto plano lo que hace más legible y fácil de depurar. Estos tienen la siguiente estructura:

.- Primero, hay una línea inicial en donde se diferencian dependiendo de si son peticiones y respuestas. Para las solicitudes la línea comienza con una acción requerida por el servidor, a esto se le denomina método de petición seguido de la url del recurso y la versión http que soporte al cliente. Lo importante es el método de petición y la URL (Uniform Resource Locator o localizador de recursos uniforme).

Para las respuestas, la línea comienza con la versión de HTTP seguido por un código de respuesta y con una frase asociada a dicho retorno.

También los mensajes tienen una cabecera que son metadatos con información diversa y el cuerpo de mensaje que es opcional. Típicamente este cuerpo tiene los datos que se intercambian entre el cliente y el servidor.

Los métodos de petición (o también llamados verbos) son varios. Cada método indica la acción que desea que se efectúe sobre el recurso identificado lo que este recurso representa dependiente de la aplicación del servidor.

Los métodos que vamos a usar son los de get y post . Cabe destacar que hay convenciones en donde se utilizan otros métodos. En este punto entran en juego el estándar REST y las API's.

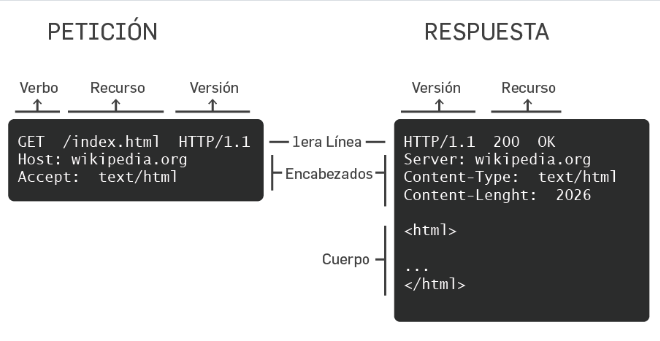
El método get solicita una representación del recurso especificado. Las solicitudes que usan sólo deben recuperar datos y no deben tener ningún otro efecto.

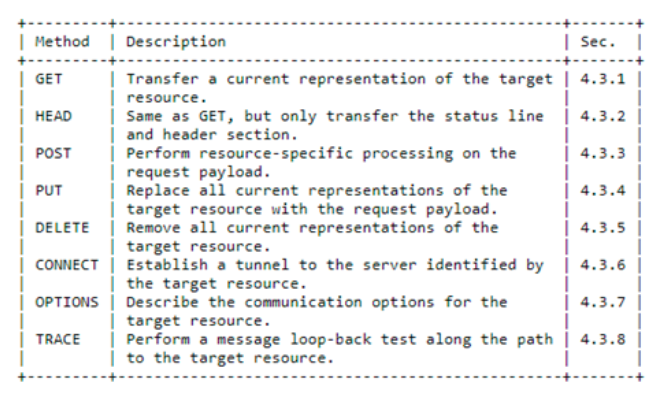
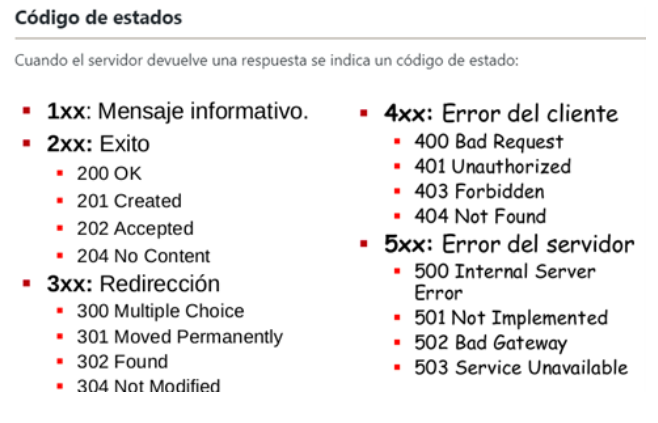
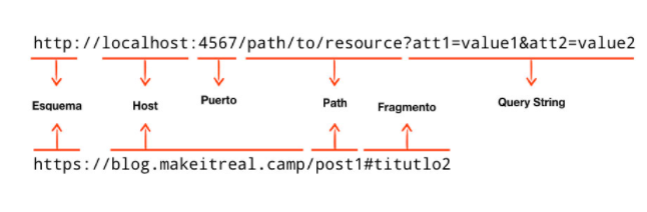
El método post envía los datos para que sean procesados ​​por el recurso identificado. Los datos enviados se incluirán en el de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes.

Para finalizar con la explicación de HTTP vamos a nombrar códigos de respuesta:

1. Por ejemplo, el 200 representa una respuesta correcta es decir que indica que la petición ha sido procesada correctamente.
2. Otro ejemplo es la respuesta 404 que significa errores causados ​​por el cliente, por ejemplo que el recurso no se encuentra.
3. Finalmente, las respuestas que comienzan con 5 por ejemplo el 500 son errores causados ​​por el servidor. Esto indica que ha habido un error en el proceso de la petición a causa de un fallo en el servidor.

Los otros protocolos de TCP / IP son necesarios para configuraciones de conexiones, pero a nivel de aplicación no los utilizaremos.





**Glosario**

***Bases de datos:*** Una base de datos es una colección organizada de información estructurada, o datos, típicamente almacenados electrónicamente en un sistema de computadora ( https://www.oracle.com/mx/database/what-is-database )

***DOM:*** El DOM ( Modelo de Objetos de Documento en español Modelo de Objetos del Documento) es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador web y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento (puede tratarse de un elemento, una cadena de texto o un comentario). ( https://developer.mozilla.org/es/docs/Glossary/DOM )

***Frameworks:*** La traducción literal de framework es ' marco de referencia ', y explica muy bien lo que significa. Un framework es un patrón o esquema que ayuda a la programación a estructurar el código ya ahorrar tiempo y esfuerzos a los programadores. ( https://fp.uoc.fje.edu/blog/que-es-un-framework-en-programacion/ )

***LAN:*** LAN (red de área local) describe una red cuya área geográfica no se extiende más de una milla ( http://www.frlp.utn.edu.ar/materias/info2/Redes%20Lan.html )

***Python:*** Python es un lenguaje de scripting independiente de plataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad ( https://desarrolloweb.com/articulos/1325.php )

***Rails:*** Ruby on Rails (o simplemente Rails) es un framework Open Source, multiplataforma y basado en el lenguaje de scripting Ruby que facilita sobremanera el diseño y desarrollo de aplicaciones web que acceden a bases de datos. Rails separa automáticamente en tres capas todos los componentes de una aplicación web (Model, View y Controller: MVC), lo que hace más sencillo y rápido el mantenimiento de aplicaciones que en otros entornos ( https://www.spri.eus/euskadinnova /es/enpresa-digitala/agenda/introduccion-ruby-rails-completo/709.aspx#:~:text=Ruby%20on%20Rails%20 )

***Swift:*** Swift es un lenguaje intuitivo de programación de código abierto creado por Apple que permite diseñar aplicaciones para iOS, Mac, el Apple TV y el Apple Watch. ( https://www.apple.com/es/swift/ ).

***WAN:*** (Wide Area Network) describe una red cuya área geográfica excede a la de una ciudad. A menudo, varias LAN o varias MAN son enlazadas para crear una WAN (http://www.frlp.utn.edu.ar/materias/info2/Redes%20Lan.html )

**Introducción gestión del tiempo**

El tiempo, en función del objetivo al que esté destinado puede dividirse en:

**Tiempo Personal:** Este es el tiempo que decidimos dedicarlo a cualquier actividad de nuestra elección personal haciendo uso de nuestra libertad de elección y de nuestro tiempo libre de compromisos.

**Tiempo Laboral:** Este es el tiempo que decidimos intercambiarlo con alguien que nos paga por hacer algún trabajo.

**Tiempo Familiar y Social:** Este es el tiempo que decidimos dedicar a las actividades sociales como fiestas, visitas a amigos, familia, etc.

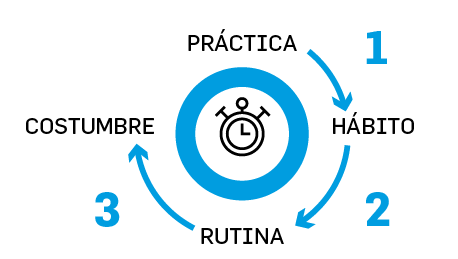
Uno de los problemas de los/as CEOs de empresas y cualquier profesional de cualquier actividad, es tener la sensación de que el tiempo del que disponemos no es suficiente para hacer todo lo que se nos presenta. Sin embargo, cada uno de nosotros disponemos de este mismo recurso, escaso y limitado, y siempre nos parece insuficiente. Además, sabemos que el tiempo es imprescindible para cualquier aspiración, deseo o actividad humana.

Sin ser tan técnicos te vamos a poner 2 posibles motivos por el cual pudiéramos tener una deficiente gestión del tiempo, pero solo ahondaremos en uno de ellos.

**Déficit cognitivo :** Este punto se refiere al funcionamiento de nuestro cerebro, es decir, el déficit es que nuestras habilidades cognitivas no son suficientes para la gestión que intentamos realizar. Un ejemplo podría ser la falta de concentración, la incapacidad para organizarse, etc. Estas funciones ejecutivas, pueden ser la habilidad de planificar el tiempo, organizar y priorizar información, revisar la ejecución, etc. que no importa si tenemos un déficit, ya que existen estrategias compensatorias para cada dificultad incorporándolas como hábitos y así poder tomar las riendas sin problemas.

**Hábitos o rutinas incorrectos :** Este punto es el más importante y debemos mirarlo con más detenimiento porque depende enteramente de nosotros observarlo, plantear una estrategia y trabajar para cambiarlo. Los hábitos o rutinas pueden ser Mentales o Físicas, pero en cualquier caso siempre es posible enfocarlas hacia objetivos planificados y a medida que va transcurriendo el tiempo se manifiestan en resultados concretos.

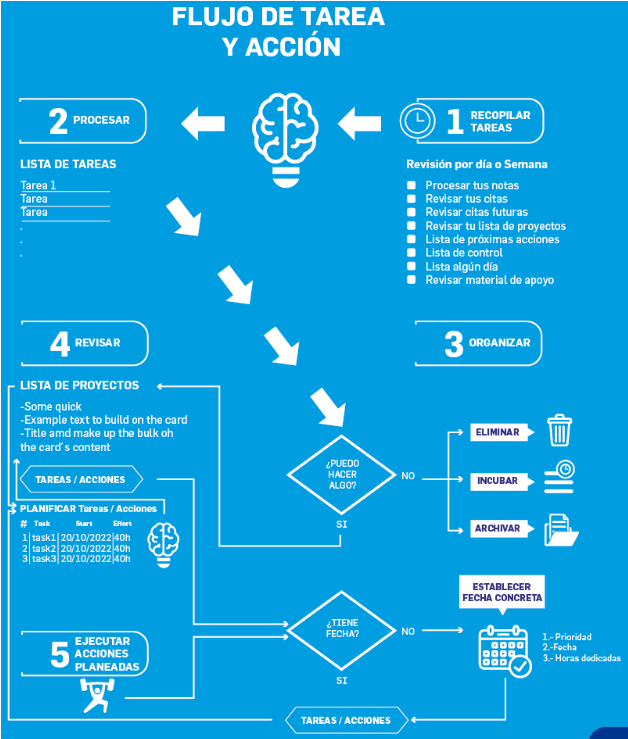
**Ciclos y hábitos en el tiempo**

**Ciclo de la práctica hacia la costumbre**

1. De la Práctica se pasa al Hábito,
2. luego del Hábito pasa a la Rutina,
3. por ultimo de la Rutina pasa a la Costumbre .

**GTD**

¿Qué es GTD? Getting Things Done (GTD) es un método de gestión de las actividades y el título de un libro de David Allen que en español se ha titulado "Organízate con eficacia". GTD se basa en el principio de que una persona necesita liberar su mente de las tareas pendientes guardándolas en un lugar específico. De este modo, no es necesario recordar lo que hay que hacer y se puede concentrar en realizar las tareas. Siempre investiga más del tema arrancando por acá .



**Time Boxing**

¿Qué es el Timeboxing? Timeboxing es una estrategia de gestión del tiempo orientada a los objetivos que puede ayudarte a aumentar la productividad y evitar la procrastinación. Cuando fijas un “bloque de tiempo”, estableces una meta para finalizar una tarea específica dentro de un período determinado.

Planificar con anticipación cuánto tiempo te llevará finalizar una tarea te permitirá organizar de manera consciente cómo usarás tu tiempo y en qué trabajarás. Esto puede ayudar a reducir los pormenores del trabajo, que actualmente ocupan el 60% de nuestro tiempo .

**1. Prioriza los bloques de tiempo establecidos**

Cuando tienes un día muy ocupado, puede resultar tentador reorganizar, reprogramar o cancelar los bloques de tiempo reservados para ese día. Trata de hacer todo lo posible para evitar eso. Una vez que hayas creado bloques de tiempo, considéralos como reuniones programadas contigo mismo.

**2. Visualiza tu tiempo para comprenderlo mejor**

El timeboxing es más efectivo cuando lo puedes visualizar. Programar el tiempo para la concentración en tu calendario te ayuda a visualizar qué límite de tiempo tienes para cada tarea en particular.

**3. Configura un temporizador**

Independientemente de si estableces un bloque de tiempo rígido o flexible, usa un temporizador para programar una alarma que te notifique cuándo se acaba el tiempo.

**4. Desactiva las notificaciones o programa el modo “No molestar”**

Los bloques de tiempo son efectivos ya que te ayudan a concentrarte en la tarea en cuestión y sumergirte de lleno en el trabajo.

**5. Tómate descansos entre bloques de tiempo**

Las estrategias de gestión del tiempo son una forma de maximizar tu eficiencia y lograr el mayor impacto posible en tu trabajo.

**6. Captura todo tu trabajo en una herramienta de gestión del trabajo**

Para crear un bloque de tiempo, primero necesitas saber en qué tienes que trabajar. Si aún no lo has hecho, crea una lista de tareas pendientes en un proyecto o en una herramienta de gestión del trabajo como Google, Trello o Asana.

**7. Agrupa las tareas similares**

Para implementar el timeboxing de manera eficaz, debes crear bloques de tiempo individuales para la mayoría de las tareas o iniciativas. agrupar tareas similares en bloques de tiempo consecutivos.

**Eficiencia vs eficacia**

**¿Cómo usas tu tiempo en el estudio o trabajo?**

Para hacer una buena gestión del tiempo es necesario tener presente, y saber la diferencia del significado de eficiencia, eficacia y el tiempo requerido para la realización de cada una de las tareas laborales, de estudios o lo que tengas que hacer.

La eficiencia consiste en hacer las cosas bien, pero no garantiza resultados. La eficiencia se centra en el método, en el cómo. Es decir, se limita a hacer bien lo que se hace.

La eficacia consiste en hacer lo que realmente hay que hacer. La persona eficaz sabe, ante todo, qué es lo que debe hacer. Tiene en mente los resultados que desea alcanzar y hace lo que le acerca a ellos. La persona que hace un buen empleo del tiempo es capaz de equilibrar eficiencia y eficacia.

**La Ley de Parkinson**

**¿En qué consiste la Ley de Parkinson?**

Un británico llamado Cyril Parkinson expuso en 1957 esta teoría tras observar su propia experiencia en una oficina pública británica. Su reflexión ayuda a entender por qué se posponen tareas y se pierde tanto el tiempo:

“El trabajo se expande hasta llenar el tiempo disponible para que se termine”, lo que viene a decir que, cuanto menos tiempo se tiene para realizar una tarea, mayor es el esfuerzo para concluirla. Mientras que cuando se tiene más tiempo, menos esfuerzo se realiza para llevarla a cabo. A esto hay que sumarle que existe una tendencia a procrastinar las tareas y a ocupar el tiempo en otras tareas que distraen del objetivo principal.

Las ideas claves que sostiene la Ley de Parkinson pueden resumirse en tres:

Trabajo: lo que comentábamos anteriormente, que el trabajo se expande hasta ocupar el tiempo disponible para realizar la tarea.

Tiempo: el tiempo que se dedica a las diferentes tareas que se realizan a lo largo del día no es proporcional a su importancia.

Gasto: el gasto aumentará hasta cubrir la totalidad de los ingresos. Esta premisa es fundamental para entender por qué muchas personas no logran llegar a fin de mes o por qué una empresa gasta todos sus recursos disponibles. Se suele decir que las necesidades de una persona se invierten en base a sus ingresos.

**¿Cómo aplicar la Ley de Parkinson en el trabajo?**

Para aplicar la Ley de Parkinson lo primero que hay que hacer es: marcarse un objetivo. De no ser así, la tarea se dilatará en el tiempo. Además, existe una norma básica que sostiene que si una tarea requiere poco tiempo para realizarla hay que hacerla de inmediato sin planificación. De esta manera, se ahorra tiempo y se reduce el número de tareas pendientes. Además, también ayudará a crear una rutina y adquirir hábitos de trabajo.

Técnica Pomodoro: basada en la filosofía del boxeo temporal, su objetivo es fijar un tiempo máximo para realizar tareas, tomar decisiones y lograr objetivos. Consiste en dividir el tiempo de trabajo en bloques de 25 minutos a los que se denominan “pomodoros”. Cuando se concluye un bloque, hay 5 minutos de descanso y tras 3 bloques de 25 se puede descansar entre 15 y 20 minutos. Durante el pomodoro no se permite ninguna distracción y en los minutos de descanso, no se permite ninguna tarea que requiera un esfuerzo mental.

A través de este método se pretende evitar el cansancio y la fatiga mental y fomentar la concentración y la agilidad mental.

Ley de Pareto o Regla 80/20: A esta regla se conoce como la Ley de Pareto. Vilfredo Pareto fue un economista y filósofo que expuso que el 80% de los resultados de una tarea proceden de hacer un esfuerzo correspondiente al 20% del tiempo dedicado. La clave de esta teoría está en concentrarse y prestar la máxima atención durante el 20% y en menor medida el 80% restante. De esta manera, se concentra un mayor esfuerzo en un período corto de tiempo y se logra diferenciar lo importante de lo que no lo es.

Evitar alargar el tiempo de trabajo: es importante ser estricto con el tiempo útil de trabajo para aprovecharlo al máximo y evitar postergar tareas. De este modo, se logra finalizar las tareas en menos tiempo y dedicándoles menos tiempo. Así se logra ser más eficiente y alcanzar las metas marcadas.

Aprovechar las horas de mayor productividad: cada persona rinde de una manera distinta y no tienen por qué coincidir en las mismas franjas horarias. Por ese motivo, es importante aprovechar las horas de mayor productividad para realizar las tareas y así lograr mejores resultados.

Evitar lo que te perjudica y anticiparse a aquellas conductas que conducen a la procrastinación, son las claves para hacer frente a la Ley de Parkinson en el trabajo o en cualquier otro ámbito. Para cumplir con los objetivos marcados se debe marcar prioridades y optimizar al máximo los espacios de tiempo que se dedican a realizar esas tareas evitando las distracciones. De esta manera, se reducirá el tiempo en llevarla a término.

**¿Cuáles son algunos principios de la eficacia?**

Principio 1: Se alargan las tareas al interrumpirlas. Las tareas se estiran y el tiempo no. Teóricamente, toda tarea requiere un tiempo determinado para obtener unos resultados -o calidad- deseados. Esta es, al menos, la impresión subjetiva que tenemos. Pero la realidad es muy distinta. Porque no nos es posible prácticamente nunca realizar de un tirón tareas medianamente complejas

Principio 2: Principio de la planificación. Cada minuto que inviertes en planificar tus actividades, te ahorra diez minutos de ejecución. El propósito de la planificación es llevarte a obtener el máximo “retorno en la inversión” que realizas en energía para tu vida personal y laboral.

Principio 3: Principio de las prioridades. Tu capacidad de fijar prioridades claras en el tiempo determina el avance hacia tus metas u objetivos.

Principio 4: Principio de las recompensas. Tus recompensas van a estar determinadas por tus resultados. La manera más efectiva de avanzar es lograr resultados de calidad en un mejor tiempo.

Principio 5: Principio de las secuencias. La administración eficaz del tiempo te permite manejar la secuencia de eventos.

**Modelo Estímulo - Respuesta**

¿Vos de qué lado estas?

Según se responde ante los estímulos que se nos plantean en la vida, se actúa de manera reactiva o proactiva:

**Reactivo:** Las personas reactivas esperan a que las cosas se arreglen solas y no son capaces de tomar decisiones; responden ante estímulos siempre de la misma manera, impulsadas por emociones, sentimientos, circunstancias, condiciones, o por el mismo ambiente.

**Proactivo:** Los proactivos son capaces de romper ese modelo porque tienen libertad interior, es decir, tienen la capacidad de elegir reaccionar ante ciertas situaciones, pudiendo así, tomar decisiones más prudentes. Afrontan la vida personal y profesional tomando la iniciativa.

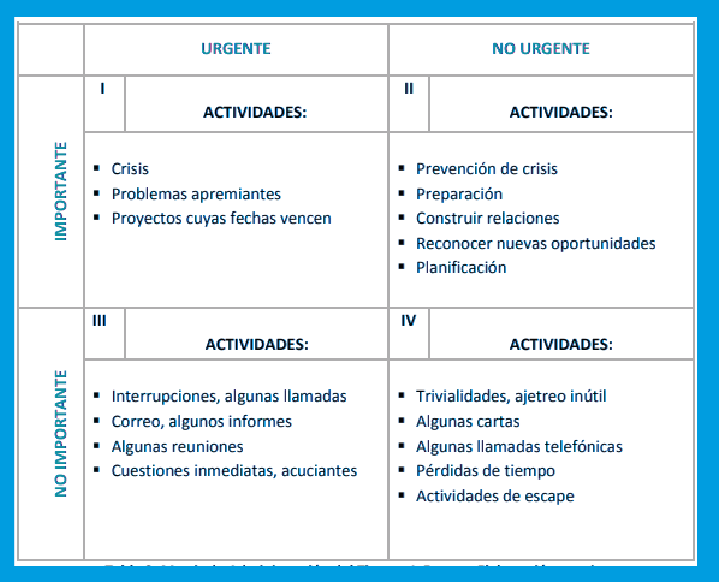
**MATRIZ DE ADMINISTRACIÓN DE TIEMPO. PRIORIZACIÓN**

La utilización eficiente del tiempo se basa en la orientación de las tareas a la consecución de los objetivos. Por lo tanto, el establecimiento de prioridades es primordial a la hora de gestionar el tiempo. Priorizando, es decir, estableciendo un orden en las tareas diarias, se puede evitar la tiranía de lo urgente y ayuda a centrarse en lo importante. Para poder establecer una clara prioridad de las actividades planificadas, se deben clasificar estas como:

● Tareas urgentes e importantes: se trata de tareas prioritarias en cuanto a que están contempladas dentro de los objetivos, pero que por falta de planificación o imprevistos de última hora, son urgentes y no se pueden demorar más.

● Tareas urgentes y no importantes: son tareas que han sido realizadas ya, pero que no son importantes puesto que su realización no aporta valor en el cumplimiento de los objetivos.

● Tareas no urgentes e importantes: son tareas contempladas como parte de los objetivos, las cuales se han sabido o pudieron planificar correctamente, con lo cual se pueden realizar con perspectiva.

● Tareas no urgentes y no importantes: son tareas que ni son importantes para el cumplimiento de los objetivos, ni son apremiantes.

**¿Qué es el desarrollo de software?**

**Desarrollo de Software**

Según la IEEE , el software es el “ conjunto de programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación ”. Es la parte de un sistema que se puede codificar para ejecutarse en una computadora como un conjunto de instrucciones, e incluye la documentación asociada necesaria para comprender, transformar y usar esa solución. Estos documentos describen la organización del sistema, explican al usuario cómo utilizarlo y obtener eventuales actualizaciones del producto.

El proceso de desarrollo de software es un conjunto de actividades que da como resultado un producto que responde a las necesidades de un usuario.

Existen varios procesos o modelos de desarrollo y podríamos catalogarlos en Secuenciales o Definidos, y en Empíricos. Los primeros responden siempre de la misma forma ante una determinada entrada, mientras que los segundos dependen de la evolución y adaptación de etapas anteriores.

**Metodología en Cascada**

Su principal característica es que cada fase del desarrollo está bien definida y separada, siguiendo un orden secuencial en el que cada etapa dependiente de la finalización de la anterior y que esta última haya pasado un proceso de validación que apruebe el paso a la siguiente.

Comúnmente las etapas del modelo son:

1. Análisis y definición de los requerimientos.
2. Diseño del sistema y del software.
3. Implementación y pruebas unitarias.
4. Integración y pruebas de sistema.
5. Funcionamiento y mantenimiento.

**Metodologia tradicional**

1. nos reunimos internamente para decidir si tomamos el proyecto,

2. invertimos tiempo y recursos en diseñar hasta el último detalle de la solución que vamos a ofrecer a la empresa (alcance del proyecto),

3. listamos y ordenamos todas las tareas que serán necesarias para construir esa solución (descomposición de la estructura de trabajo)

4. estimamos cuánto tiempo y recursos implicará la realización de esas tareas (cronograma y presupuesto del proyecto). Tomamos como base para esta estimación los proyectos similares que hicimos en el pasado.

Una vez que plasmamos toda esta información en documentos (planificación), se lo presentamos a la empresa y, si logramos un acuerdo, entonces sí:

5. Ejecutamos las tareas planificadas (ejecución)

6. Monitoreamos el progreso para evitar desvíos respecto de lo que habíamos planificado en los documentos (monitoreo y evaluación)

7. Completamos el desarrollo del software, se lo entregamos a nuestro cliente y ¡finalizamos el proyecto! (cierre).

**Metodología iterativa e incremental**

La metodología iterativa e incremental deriva del proceso de desarrollo en cascada pero, con la diferencia de que aquí se admite que las etapas se solapan en tiempo con la finalidad de flexibilizar el tiempo de desarrollo total y así poder alcanzar resultados funcionales de manera temprana.

Esta metodología involucra dos procesos fundamentales:

**El proceso incremental :** con esto se busca desarrollar una parte del producto que se pueda integrar al conjunto a medida que se alcanza un grado de completitud.

**El proceso iterativo :** se realiza en ciclos donde se revisa y mejora el producto. De manera que la calidad del producto aumente, y no siempre implica la integración de nuevas funcionalidades.

**Procesos Ágiles**

A fin de superar las dificultades o límites de las metodologías tradicionales, surgieron las metodologías ágiles para la gestión de proyectos. Los métodos o procesos ágiles consideran un enfoque iterativo para las etapas de especificación, desarrollo y entrega del software.

Estos procesos se basan en los valores y los principios definidos en el Manifiesto Ágil.

Los valores que propone este manifiesto son:

● Valorar más a los individuos y sus interacciones que a los procesos y las herramientas

● Valorar más el software funcionando que la documentación exhaustiva

● Valorar más la colaboración con el cliente que la negociación contractual

● Valorar más la respuesta ante el cambio que seguir un plan

También se ven los siguientes 12 principios:

1. Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.
2. Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
3. Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los períodos breves.
4. Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
5. Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
6. La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
7. El software que funciona es la principal medida del progreso.
8. Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica enaltece la agilidad.
10. La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan.
12. En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

Si bien hay diferentes métodos ágiles, todos comparten algunos principios elementales :

**Participación del cliente:** Los clientes deben estar implicados en todo el proceso de desarrollo, su rol está enfocado en proporcionar y dar prioridad a nuevos requerimientos del sistema, como así también participar en la evaluación de los entregables de cada iteración.

**Entrega incremental:** El software se desarrolla en incrementos y el cliente es quien determina qué necesita incluir en cada incremento.

**Enfocado en personas, no en procesos:** Hay que identificar y explotar las habilidades de cada una de las personas del equipo. Cada equipo debe desarrollar sus propias formas de trabajar, es decir, sin imponer procesos formales.

**Aceptar el cambio :** Siempre se debe tener en cuenta que los requerimientos del sistema pueden cambiar, y por lo tanto el diseño debe contemplar esto.

**Mantener la simplicidad :** Se debe enfocar la simplicidad tanto en el software que se desarrolla como en el proceso de desarrollo.

**SCRUM**

Es especialmente útil para trabajar en entornos complejos en los cuales tanto la necesidad a cubrir como la solución que la satisfaga no son claras de antemano y sólo pueden conocerse a medida que se construye y se pone a prueba el software.

Scrum es un marco de trabajo a través del cual las personas pueden abordar problemas complejos adaptativos, a la vez que se entregan productos de forma eficiente y creativa con el máximo valor. En scrum se aplica un conjunto de buenas prácticas para trabajar de manera colaborativa y obtener así el mejor resultado posible en proyectos complejos que demandan constantes cambios y adaptaciones.

Uno de los pilares de Scrum es la transparencia. Los aspectos significativos del proceso deben ser visibles para aquellos que son responsables del resultado. Esto requiere que dichos aspectos sean definidos en base a un estándar común, de tal modo que los observadores compartan un entendimiento común de lo que están viendo.

Otro de los pilares es la inspección. Los usuarios de Scrum deben inspeccionar frecuentemente los artefactos de Scrum y el progreso hacia un objetivo para detectar variaciones indeseadas. Su inspección no debe ser tan frecuente como para que pueda interferir en el trabajo. Las inspecciones son más beneficiosas cuando se realizan de forma diligente por inspectores expertos en el mismo lugar de trabajo.

Finalmente, otro pilar de la metodología es su adaptación. Si un inspector determina que uno o más aspectos de un proceso se desvían de los límites aceptables y que el producto resulta inaceptable, el proceso o el material que está siendo procesado deben ajustarse. Dicho ajuste deberá realizase cuanto antes para minimizar desviaciones mayores.

Desde la perspectiva de SCRUM, un proyecto se considera exitoso si entrega el mayor valor posible para el usuario en un tiempo dado. ¿Cómo sabemos qué aporta valor al o los usuarios? A medida que se va construyendo y testeando el software se va comprendiendo más profundamente la problemática y la necesidad de los usuarios y las posibles formas de satisfacerlo.

Los proyectos se llevan a cabo en equipos que trabajan en ciclos temporales cortos y duración fija. Cada ciclo se conoce como Sprint y tiene que proporcionar un resultado completo, un incremento del producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando este lo solicite. Los componentes del ciclo son:

● Eventos: Planificación del sprint, Scrum diario, Revisión del sprint, y Retrospectiva del sprint

● Roles: Desarrollador, Propietario del producto, y Scrum master

● Artefactos: Pila del producto, Pila del sprint, e Incremento.

**Eventos - Planificación en Scrum**

El núcleo de esta metodología son los Eventos. Cada uno de estos, se enmarca en el ciclo iterativo denominado Sprint y hace que sea posible llegar a un entregable útil al final de cada iteración. Los Sprints son el corazón de la metodología, son períodos de tiempo fijos, generalmente de un mes o menos, y comienzan inmediatamente después de finalizado el Sprint anterior. Durante un Sprint no se pueden hacer cambios que perjudiquen los objetivos del mismo, se refina la pila de producto y eventualmente se podría clarificar o renegociar el alcance con el propietario del producto a medida que se aprende sobre el avance. Solo si el objetivo se vuelve obsoleto, el propietario del producto tiene la autoridad para cancelar el Sprint .

**Planificación de Sprint**

La planificación es el evento que da inicio al Sprint y es donde se lleva a cabo la organización del trabajo a realizar. Este evento no es necesariamente exclusivo del equipo de Scrum , sino que se puede invitar a otras personas para que aconsejen en los temas pendientes.

Durante la planificación se tratan los siguientes temas:

● ¿Por qué es valioso este Sprint ?

El propietario del producto propone cómo el va a aumentar su valor y utilidad con el Sprint . Todo el equipo colabora para definir el objetivo del Sprint que transmite el por qué es valioso para los interesados ​​en el producto. Al final de este evento el objetivo debe estar bien definido.

● ¿Qué se puede hacer en este Sprint ?

A lo largo de la discusión, los desarrolladores eligen ítems de la pila de producto, para incluirlos en el Sprint en curso. Durante este proceso, el equipo puede refinar estos ítems mejorando el entendimiento sobre cada uno. Este tema, de conocer cuánto se puede hacer, los desarrolladores lo entrenan teniendo en cuenta su rendimiento pasado, la capacidad y su definición de Terminado.

● ¿Cómo se llevará a cabo el trabajo seleccionado?

Para cada uno de los elementos seleccionados, los desarrolladores planean el trabajo necesario para crear un incremento que cumpla con la definición de Terminado. La manera en que estos lo ejecutan queda a su discreción y suelen descomponer los ítems de la pila de producto en tareas más pequeñas, de un día o menos.

**Scrum Diaria**

Las Scrum diarias (Daily Scrum) son reuniones de no más de 15 minutos, en las cuales se sincroniza el trabajo desarrollado y se establece el plan para el día en curso. Generalmente se recomienda hacerlas viendo el tablero con la pila de sprint. Se comparten los avances, los problemas y las ideas, también si es necesario se actualiza la pila de Sprint . Es importante tener en cuenta que no es una reunión de inspección o control, sino que de comunicación entre el equipo.

**Revisión de Sprint**

Sobre el final del Sprint se hacen las reuniones de revisión de Sprint para comprobar el incremento. Estas suelen durar entre una y dos horas, en caso de que el incremento tenga una complejidad muy alta, podrían extenderse aún más. Estas pueden permitir al equipo y al propietario del producto tomar sensibilidad del ritmo de construcción y la trayectoria que cobra la visión del producto. Son instancias en las que se reajusta la pila de producto para incluir nuevas oportunidades.

**Retrospectiva de Sprint**

El propósito de las reuniones de Retrospectiva de Sprint es planear nuevas formas de mejorar la calidad y efectividad. Se analiza lo que sucedió en la última Sprint con respecto a los individuos, sus interacciones, los procesos, lo que fue bien y los problemas que se encontraron, los que se resolvieron y los que quedaron pendientes.

**Roles - equipo SCRUM**

**Roles / Equipo**

La unidad fundamental de Scrum es un equipo de personas (Scrum Team). Cada equipo consiste en un Scrum Master, un propietario del producto (Product Owner) y desarrolladores (Developers).

El equipo debe ser suficientemente pequeño como para ser ágil y grande como para completar una cantidad considerable de trabajo dentro de un Sprint. Normalmente los equipos están conformados por 10 personas o menos, ya que en general, los equipos pequeños se comunican mejor y son más productivos.

El equipo es responsable de todas las actividades relacionadas al producto desde la colaboración con las partes interesadas, verificación, mantenimiento, experimentación, desarrollo y cualquier otra cosa que pueda ser requerida.

**Desarrolladores**

Los desarrolladores son aquellos que producen cada uno de los incrementos de cada sprint y siempre son responsables de:

● Crear un plan para el Sprint , la pila de Sprint (Sprint Backlog).

● Asegurar la calidad mediante la adopción de una Definición de Terminado.

● Adaptar el Plan de Cada Día para conseguir el Objetivo del Sprint (Sprint Meta).

Propietario del Producto

El Propietario del Producto es el responsable de Gestionar Efectivamente la Pila de Producto (Product Backlog), Lo que incluye:

● Desarrollar y Comunicar el objetivo m del Producto (Meta del producto).

● Crear y comunicar cada uno de los artículos de la pila de producto.

● Ordenar los artículos de la pila de producto siguiendo alguna prioridad.

● Asegurar que la pila de producto sea transparente, visible y comprensible.

Es importante resaltar que el propietario del producto es una única persona, y por tanto, se deben respetar sus decisiones. Estas, deben ser reflejadas en el contenido y orden en la pila de producto, como así también en el incremento y la revisión del Sprint .

**Scrum Master**

El Scrum Master es el encargado de asegurar que la metodología se lleve a cabo tal y como está definida. Esto se logra haciendo que cada persona del equipo entienda la teoría y práctica de la metodología.

La persona con el rol de Scrum Master sirve al equipo de Scrum de varias formas:

● Orienta a los miembros del equipo en la autogestión y la multifuncionalidad.

● Ayuda a los equipos a enfocarse en crear incrementos de alto valor que cumplan con la definición de Terminado.

● Remueve los impedimentos de progreso que pudiera tener el equipo.

● Asegura que todos los eventos de Scrum se lleven a cabo y que sean positivos, productivos y dentro de los tiempos establecidos.

El Scrum Master también colabora con el propietario del producto de las siguientes maneras:

● Ayuda a encontrar técnicas para definir efectivamente el objetivo del producto y la gestión de la pila de producto.

● Ayuda a que los miembros del equipo entiendan los artículos de la pila de producto.

● Ayuda a establecer un planeamiento del producto de manera empírica para entornos que sean complejos.

Finalmente también colabora con el resto de la organización con actividades como:

● Liderar, planear y entrenar a los miembros de la organización en lo que respeta a la adopción de Scrum .

● Planear y aconsejar cómo implementar Scrum dentro de la organización.

● Ayudar a los empleados a comprender y llevar a cabo la práctica de la metodología, mediante un enfoque empírico para trabajos complejos.

Durante la reunión de planificación de Sprint, el Scrum Master deberá desempeñar las siguientes funciones:

● Realizar esta reunión antes de cada Sprint .

● Asegurar que se cuenta con una pila de producto preparada por el propietario del producto.

● Ayudar a mantener el diálogo entre el propietario del producto y los desarrolladores.

● Asegurar que se llegue a un acuerdo entre el propietario del producto y los desarrolladores con respecto a lo que incluirá el incremento.

● Ayudar a comprender la visión y las necesidades de negocio del cliente.

● Asegurar que se ha realizado una descomposición y estimación del trabajo realistas.

● Asegurar que al final de la reunión estén determinados los siguientes puntos:

1. Los ítems de la pila de producto que se van a ejecutar.
2. El objetivo del Sprint.
3. La pila de Sprint con todas las tareas estimadas.
4. La duración del Sprint y la fecha de reunión de revisión.
5. La definición de Terminado que determinará que el incremento está listo.

**Artefactos SCRUM**

**Artefactos**

**Pila de Producto**

La Pila de Producto (Product Backlog) es una lista ordenada. Cada una de las entradas de esta lista son posibles trabajos a seleccionarse para su realización durante una reunión de planificación de Sprint . El refinamiento de los ítems de la pila de producto es el acto de convertir esos ítems en elementos más detallados y precisos, en este proceso también se les asigna una prioridad y un “tamaño”.

La pila de producto puede incluir ítems para explorar las necesidades del cliente, analizar opciones técnicas, y otros ítems de trabajo tales como la corrección de errores (bugs) o la configuración del entorno. Todo lo que esté en la pila de producto representa o aporta a conseguir el objetivo del producto (Product Goal) y este último es el objetivo que el equipo tiene que conseguir cumplir a largo plazo.

**Pila de Sprint**

Este artefacto, pila de Sprint (Sprint Backlog), delimita el trabajo necesario para alcanzar determinado incremento, y sirve para marcar el avance. También sirve como herramienta de comunicación del equipo. Esto es visible para todos y representa una imagen en tiempo real del trabajo que llevan los desarrolladores para alcanzar el Objetivo del Sprint (Sprint Meta). Este último se crea durante la reunión de planificación de Sprint y a lo largo de todo el ciclo, los desarrolladores lo tienen presente.

**Incremento**

Un incremento de producto es un paso concreto que acerca el desarrollo al objetivo del producto. Cada incremento agrega valor al incremento anterior y bajo un proceso de verificación, se asegura que todos los incrementos trabajen bien juntos.

Existe un compromiso entre el incremento y la Definición de Terminado (Definición de Hecho). Esta definición es una descripción formal del estado que debe alcanzar el incremento para cumplir con el nivel de calidad requerido para el producto. Si por alguna razón algún ítem de la pila de producto no cumple con la definición de terminado, no debe formar parte del entregable y ni siquiera ser presentado en la reunión de revisión de Sprint. En lugar de esto, debe volver a la pila de producto para ser considerado en un futuro.

**Especificaciones y requerimientos**

Es importante realizar una especificación de los requisitos, es decir documental de forma completa, precisa y verificable los requisitos, el diseño y el comportamiento u otras características de un sistema o componentes del mismo.

**Requerimientos Funcionales**

Los requisitos funcionales son declaraciones de los servicios que deben proporcionar el sistema. Describen cómo debe reaccionar el sistema a entradas particulares o cómo debe comportarse bajo determinadas condiciones.

Por tanto, la especificación de requerimientos debería cumplir las características de ser completa y consistente. Para que sea completa, todos los servicios descritos por el usuario deben estar definidos. Por otra parte, para que sea consistente los requerimientos no deben ser contradictorios.

**Requerimientos no Funcionales**

Los requisitos no funcionales son aquellas propiedades que deben tener, como fiabilidad, capacidad de almacenamiento, tiempo de respuesta, etc.

Generalmente surgen de necesidades del usuario que políticas tienen que ver con restricciones de presupuesto, la interoperabilidad con otros sistemas, factores externos, regulaciones, privacidad, seguridad, etc.

**Requerimientos en Scrum y Estimación de tiempos**

Dentro de las Metodologías Ágiles se Suelen utilizar las Historias de Usuario ( User Stories ) como Herramienta para definir los Requerimientos del Sistema. Estas son descripciones o especificaciones de una función, validadas por un usuario o cliente del sistema.

Generalmente las historias se escriben en un lenguaje que el usuario pueda entender y que refleje una descripción sintetizada de lo que este desea. En lo posible se debe tratar de eliminar ambigüedades y malas interpretaciones.

Escribir las historias de usuario con este formato tiene ciertas ventajas:

Primera persona: Invita a quien la lee a ponerse en el lugar del usuario.

Priorización: Ayuda al propietario del producto a priorizar y visualizar mejor cuál es la función, quién se beneficia y cuál es el valor.

Propósito: la presencia del propósito de una función brinda al equipo la posibilidad de plantear alternativas que cumplan con el mismo objetivo en caso de que el costo de la función sea demasiado alto o su construcción no sea viable.

**Temas, Épicas e Historias**

Durante el proceso de análisis de historias de requerimientos se suele agrupar a las últimas de usuario en épicas ya su vez, estas últimas están relacionadas con un tema en particular.

Usualmente las historias de usuario se ubican en la pila del producto, son derivadas de los temas y agrupadas en épicas, luego se traducen en tareas en la pila de Sprint .

**Método INVEST**

Este método, desarrollado por Bill Wake en 2003, permite asegurar la calidad de la escritura de las historias de usuario. Para esto debe cumplir con las siguientes características:

● Independiente: Cada historia de usuario puede ser planificada e implementada en cualquier orden. No depende unas de otras, si esto ocurre se deben dividir o combinar.

● Negociable : Las historias deben ser negociables y sus detalles serán acordados por el cliente / usuario y el equipo durante la fase de «conversación».

● Valiosa: Una historia de usuario tiene que ser valiosa para el cliente o el usuario.

● Estimable: Una buena historia de usuario debe ser estimada con la precisión suficiente para ayudar al cliente o usuario a priorizar y planificar su implementación. Si no podemos estimarla debemos incidir en la fase de conversación o dividirla.

● Pequeña ( Small ): Pequeñas. Solemos hacerlas de tal modo que ocupen como máximo un sprint.

● Comprobable ( Testable ): La historia de usuario debe poderse probar (Hemos trabajado con anterioridad en la fase "confirmación" de la historia de usuario. Tanto el usuario como el equipo de desarrollo tienen que poder probarla para saber cuándo esta finalizada.

**Método SMART**

Cuando se descomponen las historias de usuario en tareas, se puede utilizar el método SMART, un método que sirve para evaluar si las tareas están correctamente definidas. SMART es un acrónimo (en inglés) de las siguientes características:

● Específico: Una tarea debe ser lo suficientemente específica para que todos puedan entenderla.

● Medible: ¿Hace lo que se pretende? Es el equipo quien debe ponerse de acuerdo sobre lo que esto significa.

● Alcanzable: ¿Es razonable la meta? Consejo: Es bueno establecer metas que representen un desafío, pero puede ser contraproducente no alcanzarlas.

● Relevante: Cada tarea debe ser relevante, contribuyendo a la historia en cuestión.

● A tiempo ( Time-Boxed ): Una tarea debe estar limitada a una duración específica. No necesariamente debe ser una estimación formal en horas o días, pero debe haber una expectativa para que la gente sepa cuándo debe ayudar.

**Subdivisión de Historias de Usuario**

Existe una guía de patrones para subdividir historias de usuario grandes o complejas en otras más pequeñas y alcanzables.

Las 3 C de las Historias de Usuario

Una historia de usuario está compuesta de tres elementos que son fundamentales. La trivial es la primera y la Tarjeta ( Card ), es Donde SE ESCRIBE La historia de Manera Clara y con la ambigüedad mínima. El segundo componente es la Conversación , es importante debatir y validar con el cliente y el equipo de desarrollo, por lo general, esto se realiza en la reunión de planificación. Y finalmente la Confirmación , es importante confirmar que todos los involucrados han comprendido los requisitos.

Puntos de historia

“El trabajo necesario para realizar un requisito o una historia de usuario no se puede prever de forma absoluta porque rara vez son realidades de una solución única. En el caso de que se pudiera, por otra parte, la complejidad de la medición haría una métrica demasiado pesada para la gestión ágil.

No resulta posible estimar con precisión la cantidad de trabajo que hay en un requisito. En consecuencia, tampoco se puede saber con antelación cuánto tiempo exigirá, porque a la incertidumbre del trabajo se suman las inherentes al tiempo: no se puede estimar la cantidad o la calidad del trabajo que realiza una «persona media» por unidad de tiempo, porque son muy grandes las diferencias de unas personas a otras. Es más: la misma tarea realizada por la misma persona requerirá diferentes tiempos según las diferentes circunstancias.

Por todas estas razones, al estimar de forma ágil, se prefiere emplear unidades relativas. En gestión ágil se suelen emplear «puntos» como unidad de trabajo, utilizando denominaciones como «puntos de historia». Cada organización, según sus circunstancias y su criterio, institucionaliza su métrica de trabajo, su «punto». Es el tamaño relativo de tareas que se suele emplear. Es importante que el significado y la forma de aplicar la métrica sea siempre la misma en las mediciones de la organización, y que sea conocida por todos. El tipo de «punto» depende de la organización. En un equipo de programación el punto puede ser equivalente a preparar una pantalla de inicio de sesión; para un equipo de diseño gráfico, la maquetación de un tríptico.

El «punto» ayuda, por un lado, a dimensionar la estimación de una tarea comparándola con una ya conocida, y por otro lado, a contrastar la dificultad que la tarea presenta para cada miembro del equipo según sus especialidades. Un ejemplo para ilustrar esto último podría ser el esfuerzo que cuesta freír un huevo. Si se estima cuántos «huevos fritos» costaría planchar una camisa, la respuesta dependiente de la persona. Alguien puede ser muy habilidoso friendo huevos, pero muy torpe para planchar camisas, y estimará que eso le costaría «8 huevos fritos»; es decir, «8 puntos». Alguien muy acostumbrado a las tareas domésticas, en cambio, podría estimar la tarea en «un punto» o «un huevo frito».

Ambos tienen razón: la cuestión es que la persona que estime sea la que va a realizar la tarea. Los «puntos de historia» suelen ser una unidad relativa o abstracta basada en algo con lo que el equipo esté muy familiarizado.

Por último, con esto se puede estimar la velocidad: en scrum, ésta es igual a la cantidad de trabajo realizado por el equipo en un sprint. Así, por ejemplo, una velocidad de 150 puntos indica que el equipo realiza 150 puntos de trabajo en cada sprint.

No obstante, al salir del marco estándar de scrum podemos encontrar sprints de diferentes duraciones. Cuando esto sucede, se puede expresar la velocidad en unidades de tiempo en lugar de por sprint. Es decir: «la velocidad media del equipo es de x puntos por semana». ”

Los puntos de historia representan un valor que sólo será relevante para el equipo de scrum, y que se usará para estimar el tamaño de una tarea (también llamada ítem, en inglés) en el backlog. Permite al equipo determinar qué tareas pueden realizar en un sprint. El product owner (“propietario del producto” en su traducción literal) podrá ver, por los puntos de la historia, qué tareas fueron relativamente complicadas de hacer por el equipo. Los puntos dan una idea del tamaño y el esfuerzo que se necesita para las tareas realizadas.

Para empezar con los puntos de la historia, el equipo debe primero definir una historia de referencia (o pivote) con la que después podrá comparar todas las historias. Se recomienda elegir una historia de usuario más o menos compleja, que incluya tantas disciplinas del equipo como sea posible. A esta historia le daremos un puntaje, posiblemente 5 u 8. Luego, se utiliza el pivote durante los perfeccionamientos del sprint backlog para determinar si otra historia de usuario es más pequeña o más grande, y qué valor puede darle.

**Planificación SCRUM**

Planificación

“La planificación de Sprint es una reunión crítica, probablemente la más importante de Scrum. Una planificación de Sprint mal ejecutada puede arruinar por completo todo el Sprint.

El propósito de la planificación de Sprint es proporcionar al equipo suficiente información como para que puedan trabajar en paz y sin interrupciones durante unas pocas semanas, y para ofrecer al Dueño de Producto suficiente confianza como para permitírselo.

Una planificación de Sprint produce, concretamente:

● Una meta de Sprint.

● Una lista de miembros (y su nivel de dedicación, si no es del 100%)

● Una Pila de Sprint (lista de historias incluidas en el Sprint)

● Una fecha concreta para la Demo del Sprint.

● Un lugar y momento definido para el Scrum Diario.

Es importante que tanto el equipo como el propietario del producto asistan a la planificación de Sprint ya que cada historia contiene tres variables que son muy dependientes unas de otras.

El alcance y la importancia los fija el propietario del producto. La estimación la proporciona el equipo. Durante una planificación de Sprint, estas variables tienen un ajuste fino y continuo a través del diálogo cara a cara entre el equipo y el propietario del producto ”.

La reunión que se realiza al comienzo de cada Sprint donde participa el equipo completo; sirve para inspeccionar el Product Backlog y que el equipo de desarrollo selecciona el Product Backlog Items en los que va a trabajar durante el siguiente Sprint. Durante esta reunión, el Propietario del producto presenta el Backlog del producto actualizado que el equipo de desarrollo se encarga de estimar, además de intentar clarificar aquellos ítems que crean necesarios.

Durante esta etapa se inspeccionan el Product Backlog, los acuerdos de la Retrospectiva, la capacidad y la Definición de Done y se adaptan el Sprint Backlog, Sprint Goal y el plan para poder alcanzar ese Sprint Goal."

Planificación de póquer

A la hora de estimar tiempos se trata de alcanzar los siguientes objetivos:

● Tener diferentes puntos de vista. En la estimación ágil se busca que todo el mundo participe y diga la suya. Si hay discusión mejor, ya que de lo que se trata es de avanzar todo lo posible en los problemas y tenerlos en cuenta desde el inicio. No debería haber personas con una voz y voto más fuertes que los demás.

● Detectar posibles tareas ocultas y posibles obstáculos. La sesión de estimación es una de las primeras oportunidades de detectar riesgos que pueden comenzar a tratarse para que no se conviertan en impedimentos.

● Tener una visión compartida del trabajo que se va a realizar. Es muy útil haber participado en las estimaciones para después hacer las planificaciones. Conocer el tamaño de las historias y tareas permite que sea más fácil comprometerse con un plan de trabajo.

● Tener estimaciones más realistas (no más precisas). Para ello necesitas eliminar la presión contractual, esto es, dejar margen para equivocarse y evitar así introducir buffers “inconscientes” por si acaso. Lo que buscamos es realismo, no precisión, es decir, quiero saber si una historia serán 3 o 5 días, si me dices que tardarás 26,5 horas dudaré de que hayas hecho un buen ejercicio de estimación.

Planning Poker es una técnica efectiva para la estimación ágil. Se reúne al equipo y se utiliza una baraja de poker modificada con la que se hacen rondas de estimación con ayuda de estas cartas.

La baraja de cartas tiene una pseudo secuencia de Fibonacci modificada, cada participante recibe cartas con valores 0, ½, 1, 2, 3, 5, 10, ? e infinito, donde el 0 significa que la historia ya está hecha o no requiere esfuerzo, el interrogante significa que falta información para estimar la tarea o historia, finalmente el infinito significa que el trabajo es demasiado grande y habría que subdividirla.

Para comenzar la sesión de estimación se suele realizar una ronda de preguntas para estimar y para despejar cualquier tipo de duda sobre las historias que se van a estimar. Luego se leen y discuten una por una las historias y cada uno de los integrantes elige una carta en función del esfuerzo que debe requerir esa historia, es importante aclarar que solo se pueden elegir valores incluidos en la baraja. Finalmente, si no hay consenso, se abre discusión donde cada uno explica su elección, luego se repite la estimación en busca de un consenso y, si al final no hay consenso, se elige la media o el valor más alto.

Esta técnica tiene varias ventajas:

● Todos los miembros del equipo expresan su opinión sin sentirse condicionados por el resto.

● Al ser consciente del esfuerzo que supone, aumenta el grado de implicación de los componentes del equipo.

● Al sentirse partícipe, el grado de compromiso con el proyecto también aumenta.

● Hay más efectividad a la hora de estimar las fechas de entrega del proyecto.

**Git Introducción**

**Sistemas de Control de Versiones**

En el proceso de desarrollo de software es un requisito casi indispensable mantener un registro de los cambios que se realizan sobre el código fuente a lo largo del tiempo. Es debido a esto que cobran importancia los sistemas de control de versiones. Estos sistemas son herramientas que permiten realizar un seguimiento de los cambios y también permitir proteger el código de errores humanos accidentales. Además, un sistema de control facilita el trabajo en equipo a la hora de desarrollar software, ya que mientras un integrante trabaja en alguna función específica, otro podría estar trabajando en alguna corrección de errores o bien en otra función, para luego integrar las soluciones y realizar una sincronización del trabajo de cada uno.

El uso de un sistema de control de versiones tiene tres ventajas principales:

Gracias al historial de cambios se puede saber el autor, la fecha y notas escritas sobre los cambios realizados. También permite volver a versiones anteriores para ayudar a analizar causas raíces de errores y es crucial cuando hay que solucionar problemas de versiones anteriores.

Creación y fusión de ramas. Al tener varios integrantes del equipo trabajando al mismo tiempo, cada uno en una tarea diferente, pueden beneficiarse de tener flujos de trabajo independientes. Posteriormente se pueden fusionar estos flujos de trabajos o ramas a una principal. Los sistemas de control de versiones tienen mecanismos para identificar que los cambios entre ramas no entren en conflicto para asegurar la funcionalidad y la integración.

Trazabilidad de los cambios que se hacen en el software. Poder conectar el sistema de control con un software de gestión de proyectos y seguimiento de errores, ayuda con el análisis de la causa raíz de los problemas y con la recopilación de información.

El concepto de versión (también llamado revisión o edición) de un proyecto (código fuente) hace referencia al estado en el que se encuentra el mismo en un momento dado de su desarrollo o modificación. Los sistemas de control de versiones utilizan repositorios para almacenar el proyecto actualizado junto a sus cambios históricos. Los sistemas de control de versiones centralizados almacenan todo el código en un único repositorio, es decir que un único servidor contiene todos los archivos versionados. Esto representa un único punto de falla dado que si el servidor no está disponible por un tiempo nadie podrá colaborar o guardar cambios en archivos en los que hayan estado trabajando.

Los sistemas de control de versiones distribuidos permiten en cambio continuar el trabajo aún cuando el repositorio de referencia no está disponible. En estos sistemas los clientes no solo descargan la última copia del código, sino que se replica completamente el repositorio con los cambios históricos (versiones). De esta manera, si un servidor deja de funcionar y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios estarán disponibles para los clientes y puede ser copiado al servidor con el fin de restaurarlo.

**Git**

Git es un proyecto de código abierto maduro y con un activo mantenimiento desarrollado originalmente por Linus Torvalds. Este sistema de control de versiones distribuido funciona bajo cualquier plataforma (Windows, MacOS, Linux, etc.) y está integrado en una amplia variedad de entornos de desarrollo (IDEs). Este sistema presenta una arquitectura distribuida, es decir que, cada desarrollador posee una copia del trabajo en un repositorio local donde puede albergar el historial completo de todos los cambios y, mediante comandos determinados, realiza sincronizaciones al repositorio remoto.

Git fue diseñado teniendo en cuenta las siguientes características:

**Rendimiento:** Los algoritmos implementados en Git aprovechan el profundo conocimiento sobre los atributos comunes de los auténticos árboles de archivos de código fuente. El formato de objeto de los archivos del repositorio de Git emplea una combinación de codificación delta (que almacena las diferencias de contenido) y compresión, guardando explícitamente el contenido de los directorios y los objetos de metadatos de las versiones.

**Seguridad:** la principal prioridad es conservar la integridad del código fuente gestionado. El contenido de los archivos y las verdaderas relaciones entre estos y los directorios, las versiones, las etiquetas y las confirmaciones, están protegidos con un algoritmo de hash criptográficamente seguro llamado "SHA1". De este modo, se salvaguarda el código y el historial de cambios frente a las modificaciones accidentales y maliciosas, garantizando que el historial sea totalmente trazable.

**Flexibilidad:** es flexible en varios aspectos, en la capacidad para varios tipos de flujos de trabajo de desarrollo no lineal, en su eficiencia en proyectos tanto grandes como pequeños y en su compatibilidad con numerosos sistemas y protocolos. Se ha ideado para posibilitar la ramificación y el etiquetado como procesos de primera importancia y las operaciones que se refieren a las ramas y las etiquetas (como la fusión o la reversión) también se almacenan en el historial de cambios.

**Áreas y estados**

Para trabajar con git es fundamental entender los estados por los que pueden pasar los archivos durante todo el flujo de desarrollo.

En un proyecto de Git hay 4 secciones fundamentales:

1. **Directorio de Trabajo de (Working Area ):** Es una copia de una versión del Proyecto, Archivos sacados de la BASE DE DATOS comprimida y se colocan en el disco para ser usados o modificados.
2. **Área de preparación ( Staging Area ):** Es un archivo que se encuentra dentro del directorio de Git y que contiene información acerca de lo que va a ir en la próxima confirmación.
3. **Directorio de Git ( Local Repository ):** Es el lugar en donde se almacenan los metadatos y la base de datos de objetos del proyecto. Es lo que se copia cuando se clona un repositorio desde otra fuente.
4. **Repositorio Remoto ( Remote Repository ) :** Es el repositorio que se encuentra en un servidor remoto y con el que eventualmente se sincroniza los trabajos entre los diferentes integrantes del equipo.

Es importante tener presente siempre en qué áreas internas de Git están nuestros archivos, porque el no saber exactamente residen los archivos puede ocurrir que perdamos cambios por confusión. El objetivo es que los archivos que ya sabemos que están funcionando correctamente queden en el Repositorio Local y en caso que se quiera compartir el código debemos enviarlo al Repositorio remoto.

Para mover los archivas a las distintas áreas dentro de Git debemos utilizar los comandos desde la consola, en la siguiente imagen te compartimos un ejemplo cómo los comandos mueven los archivos a distintas áreas.

**Comandos Básicos**

Git se puede ver como un set de herramientas muy completo, pero para un manejo básico de repositorios en Git es necesario conocer por lo menos, los siguientes comandos:

* **git init** es el comando para inicializar un directorio como repositorio Git, se ejecuta dentro del directorio del proyecto, y como resultado crea un subdirectorio ***.git*** que contiene todos los archivos para poder realizar el seguimiento de los cambios, etiquetas, etc.
* **git add <archivo**> luego de la creación, modificación o eliminación de un archivo, los cambios quedan únicamente en el área de trabajo, por lo tanto es necesario pasarlos al área de preparación mediante el uso del comando ***git add*** , para que sea incluido dentro de la siguiente Confirmación ( cometer ).
* **git status** es un comando que permite conocer en qué estado se encuentran los archivos
* **git commit**, con este comando se confirman todos los cambios registrados en el área de preparación, o lo que es lo mismo, se pasan los cambios al repositorio local.
* **git push** es el comando que se utiliza para enviar todas las confirmaciones registradas en el repositorio local a un repositorio remoto.
* **git pull** funciona al inverso de ***git push***, trayendo todos los cambios al repositorio local, pero también dejándolos disponibles directamente para su modificación o revisión en el área de trabajo. Es importante mencionar que se utiliza cuando ya se tiene un repositorio local vinculado a uno remoto, al igual que con el comando ***git push***.
* **git clone** , en el caso de necesitar "bajar" un repositorio remoto de algún proyecto ya existente se puede ejecutar este comando. Genera un directorio (con el nombre del repositorio o uno especificado explícitamente) que contiene todo lo propio al proyecto, además del subdirectorio ***.git*** necesario para poder gestionar los cambios y todo lo pertinente al repositorio Git.

**Github**

GitHub es una plataforma de colaboración formal e informal de desarrollo de software (conocida también como plataforma de social coding). en esta se pueden publicar repositorios remotos que funcionan bajo el sistema de control de versiones Git. La plataforma configura los proyectos nuevos como de código abierto, por lo que cualquier persona puede verlos, pero esto es configurable.