

**UNIVERSIDAD MAYOR DE SAN ANDRES
FACULTAD DE CIENCIAS PURAS Y NATURALES
CARRERA DE INFORMATICA**

Materia: Ingenieria de Software Inf-163

Docente: Ph.D. Luisa Velasquez Lopez

Participantes: Blanco Salazar Luis Alberto

Cori Mamani Juan Wilson

Gomez Paillo Edwin Eduardo

Gutierrez Huañapaco Ivan Israel

Limachi Lopez Adalid Osmar

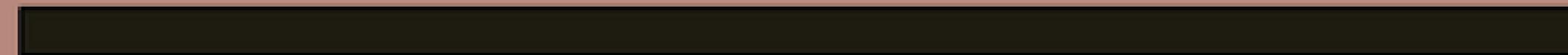
Osco Yanapatzi Mariel Karime

mantengamos las divisiones al final lo quitaremos



1 LÍNEA DE TIEMPO DEL DESARROLLO DE SOFTWARE

UNIV. JUAN WILSON CORI MAMANI



La primera teoría sobre el software fue propuesta por Alan [Turing](#) en su ensayo de 1935 sobre números computables, con una aplicación destinada a la toma de decisiones. El término "software" fue utilizado por primera vez de forma escrita por John W. [Tukey](#) en 1958.

La línea de tiempo del desarrollo de software es más clara desde 1950, sin embargo desde inicios de 1940, escribir software ha evolucionado hasta convertirse en una profesión que se ocupa de cómo crear software y maximizar su calidad.



LINEA DE TIEMPO DEL DESARROLLO DE SOFTWARE



Primera Etapa Cronológica (1950-1960):
"Programación o técnicas de codificación"

Segunda Etapa Cronológica
(1960-1970): "Modelo de procesos"

Tercera Etapa Cronológica (1970-1985):
"Proceso de Desarrollo Software y
Modelos Tradicionales del Ciclo de Vida"

Quinta Etapa Cronológica (2000 al
presente): "Metodologías del Proceso de la
Ingeniería de Software"

Cuarta Etapa Cronológica (1985-1999):
"Métodos Rápidos e Inicios del Desarrollo
Ágil de la Ingeniería de Software"

Primera Etapa Cronológica (1950-1960): “Programación o técnicas de codificación”

En esta época no existían metodologías de desarrollo. Las personas que desarrollaban los sistemas eran programadores más enfocados en la tarea de codificar, que en la de recoger y comprender las necesidades de los usuarios.

COBOL

LANGUAGE

hero[®]
Fortran

REPORTS REPORTS



Primera Etapa Cronológica (1950-1960): “Programación o técnicas de codificación”

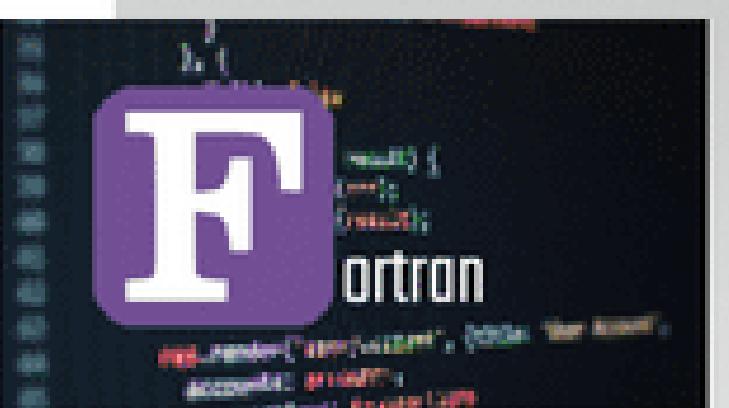
A medida que la complejidad de las tareas que realizaban las computadoras aumentaba, se hizo necesario disponer de un método sencillo para programar. Entonces, se crearon los lenguajes de programación de tercera generación que diferían de las generaciones anteriores (lenguaje ensamblador conocido como segunda generación y lenguaje de máquina como primera generación) en que sus instrucciones o primitivas eran de alto nivel (comprendibles por el programador, como si fueran lenguajes naturales) e independientes de la máquina.





Primera Etapa Cronológica (1950-1960): “Programación o técnicas de codificación”

Estos lenguajes se llamaron lenguajes de alto nivel. Los ejemplos más conocidos son FORTRAN (FORmula TRANslator) que fue desarrollado para aplicaciones científicas y de ingeniería, y COBOL (COmmon Business-Oriented Language), que fue desarrollado por la U.S. Navy de Estados Unidos, para aplicaciones de gestión o administración



Segunda Etapa Cronológica (1960-1970): “Modelo de procesos”

Desde que se empezó a trabajar sobre el desarrollo de programas, se siguieron ciertos métodos que permitían llevar a producir un buen proyecto, estas metodologías aplicadas eran simples, solo se preocupaban por los procesos mas no por los datos, por lo tanto los métodos eran desarrollados hacia los procesos. El modelo de procesos predominaba para los años 60 y consistía en codificar y corregir (Code-and-Fix).

Segunda Etapa Cronológica (1960-1970): “Modelo de procesos”

A pesar que el desarrollo de software tomó una connotación de tarea unipersonal y donde el programador era el usuario de la aplicación, se lo consideró como una base inicial para la fabricación del software, en vista de que en este modelo se empieza a establecer una idea general de lo que se necesita construir, se utiliza cualquier combinación de diseño, código, depuración y métodos de prueba no formales que se los aplica hasta que se tiene el producto listo para entregarlo.

Segunda Etapa Cronológica (1960-1970): “Modelo de procesos”

Si al terminar se descubría que el diseño era incorrecto, la solución era desecharlo y volver a empezar. Este modelo implementaba el código y luego se pensaba en los requisitos, diseño, validación y mantenimiento.



Segunda Etapa Cronológica (1960-1970): “Modelo de procesos”

Paralelamente, aparece la Crisis del Software, llamada así por los problemas que surgieron a medida que se daba el desarrollo del software. Especialmente fue marcada por los excesos de costos, la escasa fiabilidad, la insatisfacción de los usuarios y el tiempo de creación de software que no finalizaba en el plazo establecido; todos estos aspectos conocidos como “síntomas” de la crisis de software.

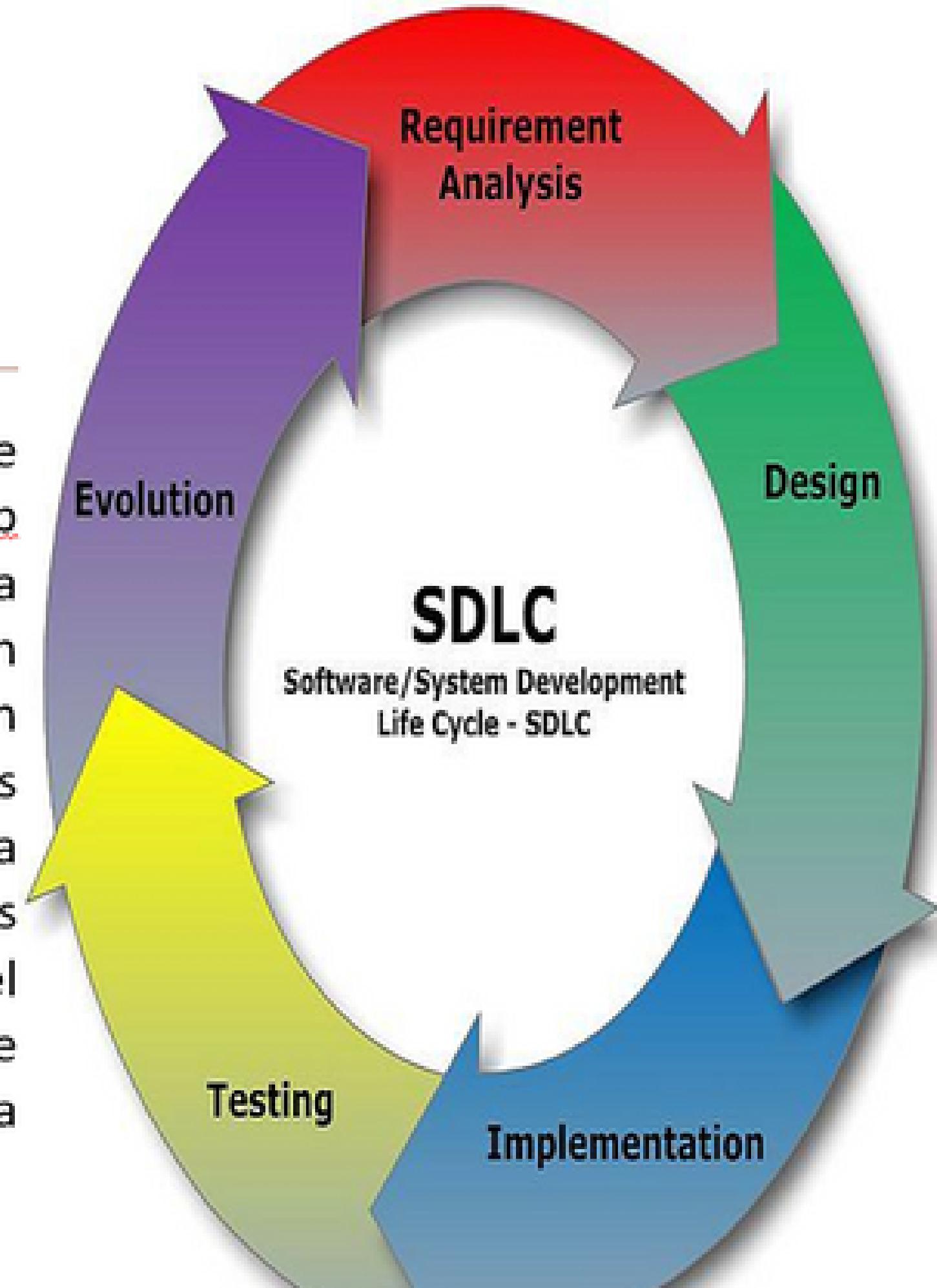
Esto provocó grandes pérdidas en la década de los 70's sobre el desarrollo de software, dando como resultado una nueva disciplina llamada “Ingeniería del Software” que abarcaría los aspectos técnicos del software y la gestión de datos.



Tercera Etapa Cronológica (1970-1985): “Proceso de Desarrollo Software y Modelos Tradicionales del Ciclo de Vida”

Esta tercera etapa está marcada por las soluciones que deben realizarse para resolver la llamada “Crisis del Software”. Mientras que el término Ingeniería del Software fue acuñado en la Conferencia de Ingeniería de Software de la OTAN en 1968 para discutir la crisis, los problemas que intentaba tratar empezaron mucho antes.

El ciclo de vida de desarrollo de software o SDLC (Software Develop Life Cicle) empezó a aparecer, a mediados de la década, como un consenso para la construcción centralizada de software, y daría las pautas en la que se logra establecer, de manera general, los estados por los que pasa el producto software desde que nace a partir de una necesidad, hasta que muere.



Tercera Etapa Cronológica (1970-1985): “Proceso de Desarrollo Software y Modelos Tradicionales del Ciclo de Vida”

A estos modelos se les denomina “Modelos de ciclo de vida del software”, los cuales pretenden abarcar todo el proceso completo creando en cada paso normativas y parámetros describiendo el desarrollo de software desde la fase inicial hasta la final y orientarlo a ajustarse a las propias necesidades de cada empresa y que sean aceptados internacionalmente.

Por otro lado, se empieza a descubrir los errores y las omisiones en los requerimientos originales del software. Surgen los errores de programa y diseño, y se detecta la necesidad de nueva funcionalidad.

A finales de esta etapa, aparece el desarrollo en espiral de Barry Boehm en 1985, el cual sería utilizado de forma generalizada en la ingeniería del software. A diferencia de otros modelos de proceso que finalizan cuando se entrega el software, el modelo se adaptaría para aplicarse a lo largo de toda la vida del software de cómputo.

Cuarta Etapa Cronológica (1985-1999): “Métodos Rápidos e inicios del Desarrollo Ágil de la Ingeniería de Software”



A mediados de los años 80, es la época marcada por la distinción y conceptualización de los métodos de la ingeniería de software cuya importancia va tomando cuerpo en las organizaciones. Se empiezan a estudiar los objetos en sí como unidades de información, dando como punta pie inicial para el proceso evolutivo de desarrollo software el modelo en espiral presenciado a finales de la etapa anterior.

Para los años 90 se quiere dar respuesta al entorno siempre cambiante y en rápida evolución en que se han de desarrollar los programas informáticos, lo cual da lugar a trabajar en ciclos cortos (como mini-proyectos) que implementan una parte de las funcionalidades, pero sin perder el rumbo general del proyecto global.

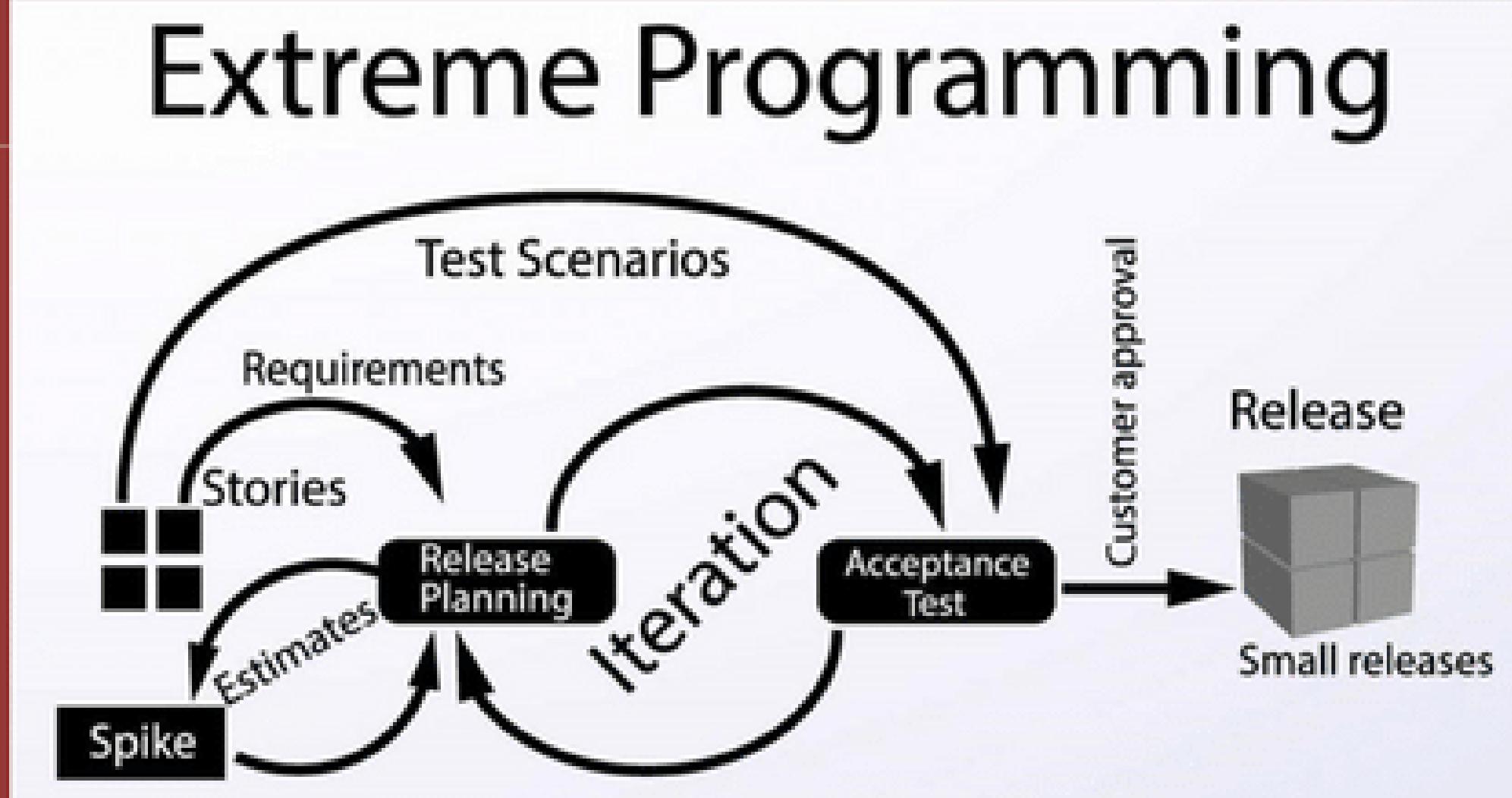
Cuarta Etapa Cronológica (1985-1999): “Métodos Rápidos e inicios del Desarrollo Ágil de la Ingeniería de Software”

Surge entonces el desarrollo de software de “métodos rápidos” (también denominado Modelo rápido o abreviado AG) los cuales reducirían el tiempo del ciclo de vida del software (por lo tanto, acelera el desarrollo) al desarrollar, en primera instancia, una versión prototipo y después integrar la funcionalidad de manera iterativa para satisfacer los requisitos del cliente y controlar todo el ciclo de desarrollo.

Empezando con las ideas de Brian Gallagher, Alex Balchin, Barry Boehm y Scott Shultz, James Martin desarrolló el enfoque de desarrollo rápido de aplicaciones durante los 80 en IBM y lo formalizó finalmente en 1991, con la publicación del libro, “Desarrollo rápido de aplicaciones”. Aparece entonces el Desarrollo rápido de aplicaciones (RAD), para responder a la necesidad de entregar sistemas muy rápido.

Cuarta Etapa Cronológica (1985-1999): “Métodos Rápidos e inicios del Desarrollo Ágil de la Ingeniería de Software”

Para 1996, surge **Extreme Programming** (XP) o Programación Extrema fundada por Ken Beck, identificando qué era lo simple y difícil al momento de programar. Centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo.





Quinta Etapa Cronológica (2000 al presente): “Metodologías del Proceso de la Ingeniería de Software”

En el año 2001, miembros prominentes de la comunidad de desarrollo software se reunieron en Snowbird, Utah, y adoptaron el nombre de “métodos ágiles”. Poco después, algunas de estas personas formaron la “alianza ágil”, una organización sin fines de lucro que promueve el desarrollo ágil de aplicaciones.

La filosofía de las metodologías ágiles, pretenden dar mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.



Quinta Etapa Cronológica (2000 al presente): “Metodologías del Proceso de la Ingeniería de Software”

Se puede apreciar claramente la evolución paulatina que ha desembocado hasta conformar la concepción de las **Metodologías del Proceso de Ingeniería de Software**, la cual describe el conjunto de herramientas, técnicas, procedimientos y soporte documental para el diseño del **Sistema Software**.

Esta etapa cronológica logra establecer metodologías que pueden involucrar prácticas tanto de Metodologías Ágiles como de Metodologías Tradicionales o Convencionales.

Tener metodologías diferentes para aplicar de acuerdo con el proyecto que se desarrolle resulta una idea interesante.

Los nuevos métodos que paulatinamente van generándose, buscan minimizar riesgos y, puesto que los errores más perjudiciales se producen en los primeros pasos, se comienza ya desde la fase más general del estudio por analizar los riesgos que significa seguir con las siguientes fases del desarrollo.

Quinta Etapa Cronológica (2000 al presente): “Metodologías del Proceso de la Ingeniería de Software”

Si los riesgos son superiores a lo que se consideran permisibles, se vuelve al paso anterior o se abandona el desarrollo. No sólo se realizan desarrollos lineales, en cascada, sino también desarrollos y métodos en espiral que son notablemente más complejos, apoyándose también con el desarrollo ágil.

Metodologías de desarrollo de software



A modo de conclusión, históricamente, las metodologías tradicionales han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes.

Una de las cualidades más destacables en una metodología ágil es su sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costes de implantación en un equipo de desarrollo.

D

TRANSICIONES

ANIMACIONES

PRESENTACIÓN CON DIPOSITIVAS

REVISAR

VISTA

H.A.

Estilos
rápidos

?

Herramienta de diagnóstico de DirectX

Sistema Pantalla Representar Sonido Entrada

Esta herramienta informa detalladamente acerca de los componentes y controladores de DirectX instalados en el sistema.

Si conoce la causa del problema, haga clic en la ficha correspondiente arriba. De lo contrario, use el botón "Página siguiente" abajo para visitar las páginas en secuencia.

Información del sistema

Fecha y hora actuales: lunes, 11 de abril de 2022, 03:05:46 p.m.

Nombre del equipo: HP

Sistema operativo: Windows 8.1 Single Language 64 bits (6.3, compilación 9600)

Idioma: español (configuración regional: español)

Fabricante del sistema: Hewlett-Packard

Modelo del sistema: HP 15 Notebook PC

BIOS: F.23

Procesador: Intel(R) Core(TM) i5-4210U CPU @ 1.70GHz (4 CPUs), ~2.4GHz

Memoria: 4028MB RAM

Archivo de paginación: 5541MB usados, 3269MB disponibles

Versión de DirectX: DirectX 11

Comprobar firmas digitales de los Laboratorios de calidad de hardware de Windows (WHQL)



DxDiag 6.0.3.9600.17415 64 bits Unicode Copyright © Microsoft. Todos los derechos reservados.

Ayuda

Página siguiente

Guardar la información...

Salir

ar para agregar notas

NOTAS

COMENTARIOS



2 LÍNEA DE TIEMPO DEL DESARROLLO DE LENGUAJES DE PROGRAMACIÓN.

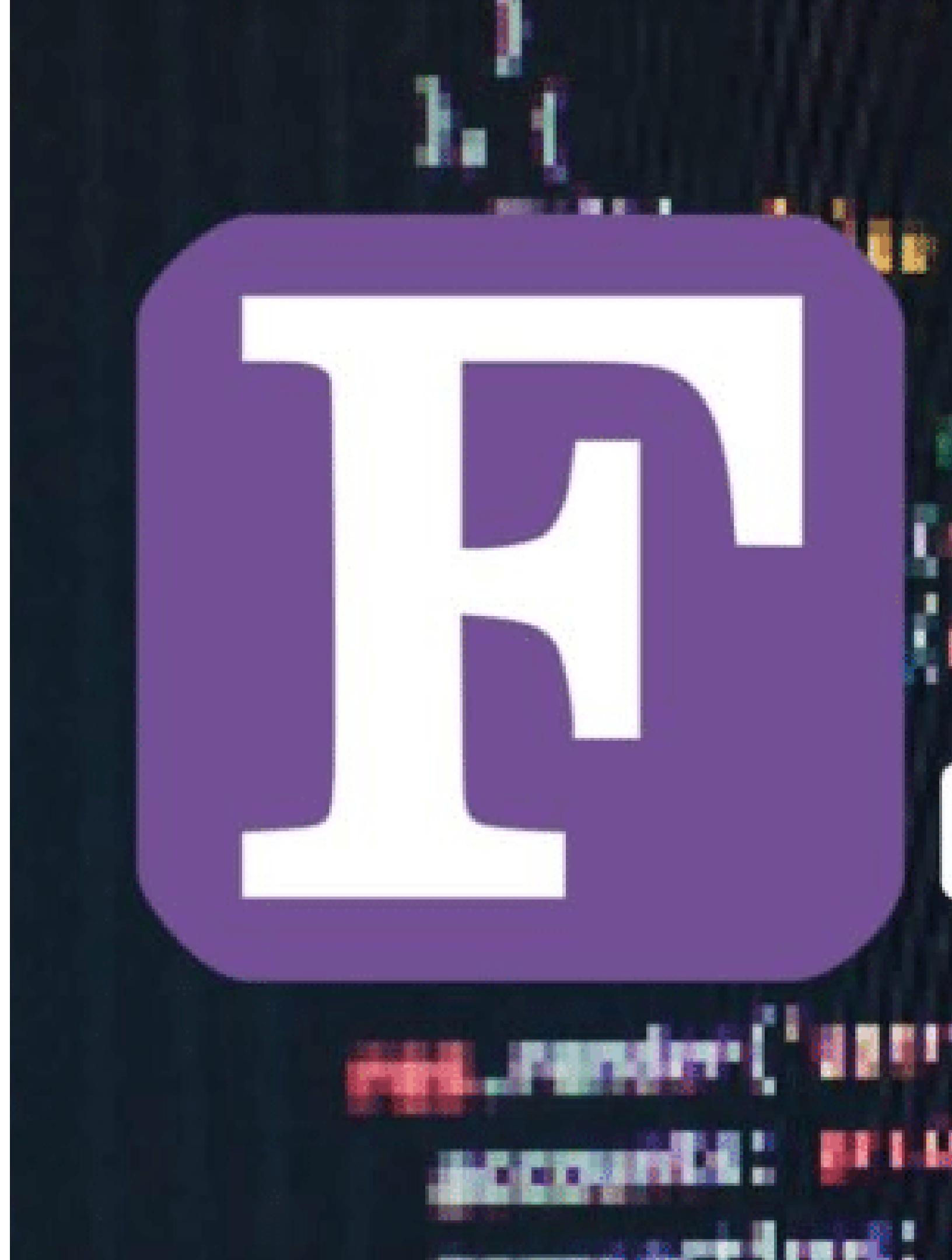
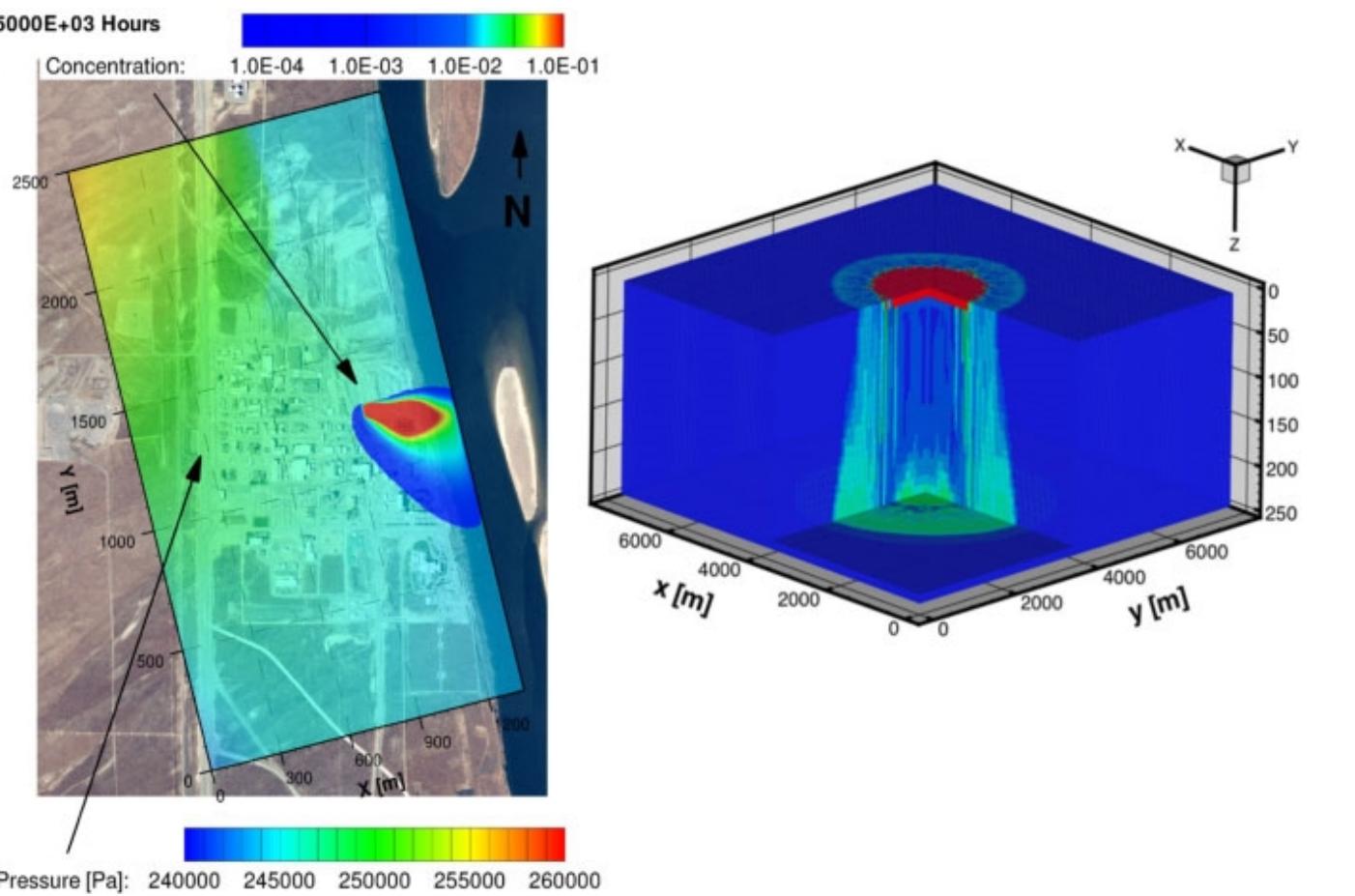
Univ. Adalid Osmar Limachi Lopez



1950 - 1965

FORTRAN

- Desarrollado originalmente por IBM en 1957
- Especialmente adaptado al cálculo numérico y a la computación científica



1950 - 1965

COBOL

- COBOL fue creado en el año 1959
- El objetivo era crear un lenguaje de programación universal y orientado principalmente a los negocios



COBOL



60 million
patients are
cared for
daily



95%
ATM transactions
daily



96%
vacations
booked daily



60 million
Lines of code
at the Social
Security
Admin



80%
Point of Sale
transactions
daily



50 million
lines of code at
the IRS



PASCAL

Programming Language

1965 - 1972

PASCAL

- PASCAL fue publicado en el año 1970
- Utilizando la programación estructurada y estructuración de datos





1965 - 1972

C

- C fue desarrollado por Dennis Ritchie entre 1969 y 1972
- Es apreciado por la eficiencia del código que produce y es el más popular para crear softwares



1972 - 1985

BASIC

- Creado por John Kemeny y Thomas Kurtz en 1974
- Escribir programas usando terminales de computador de tiempo compartido



```
70 B$$="WORLD!"  
80 A$$=BS$  
90 X=RND(0)  
100 X=SIN(X)  
110 PRINT I;" ";H  
120 X=RND(0)  
130 X=COS(X)  
140 X=EXP(1)  
150 X=LOG(X)  
160 B=VAL("100")  
170 C=LEN(A$)  
180 X=ABS(X)  
190 IF X<1000 THEN  
195 FOR J=1 TO 5  
196 Y=1000*SRK(1)  
197 NEXT J  
200 NEXT I  
210 GOTO 500  
220 X=0  
230 RETURN  
240 PRINT "KONEC"
```

1972 - 1985

C++

- Diseñado por los años 80 por el Bjarne Stroustrup
 - Su intención fue la de extender el lenguaje de programación C





1985 - 2000

JAVA

- Comercializada en 1995 por Sun Microsystems
- Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado





1985 - 2000

JAVASCRIPT

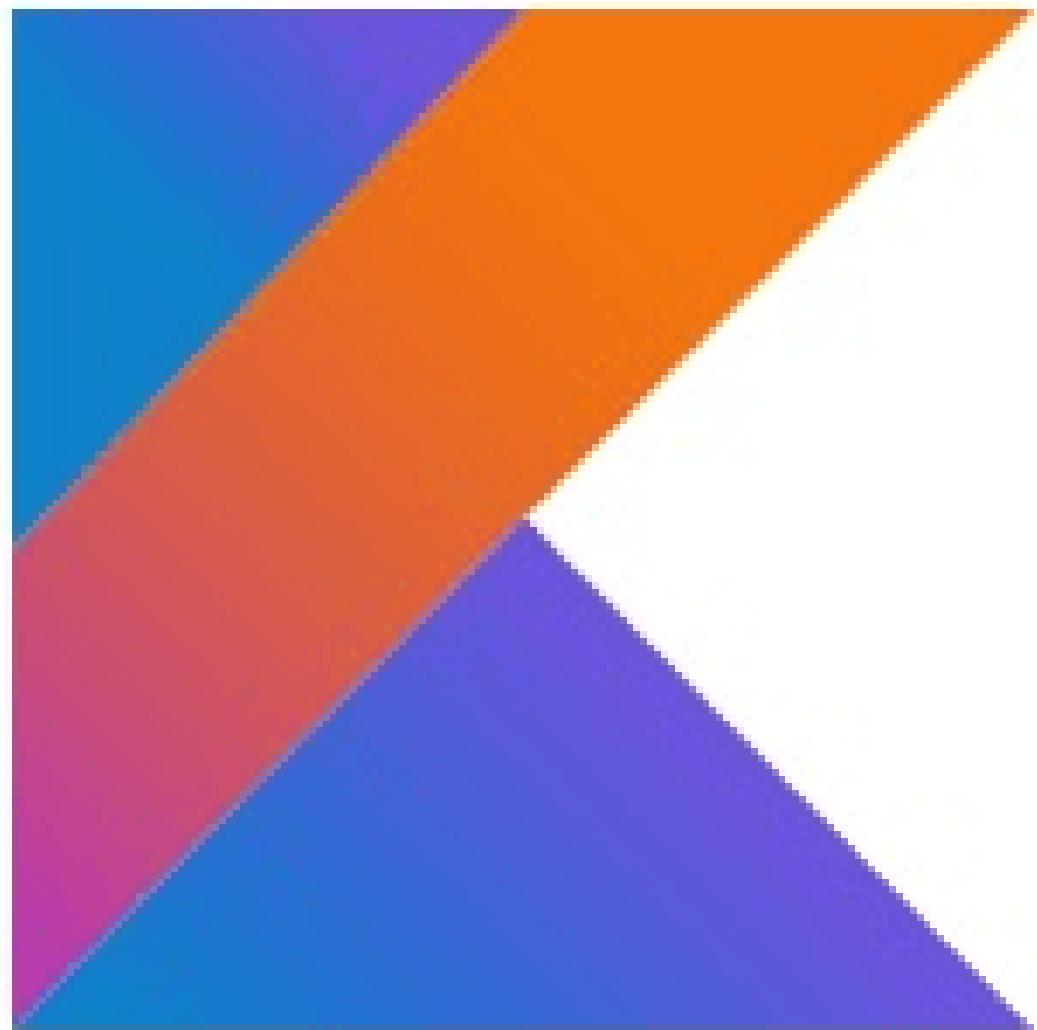
- Desarrollado por Brendan Eich de Netscape
- Muchas librerías que están basadas en javascript



2000 - Presente

KOTLIN

- Revelado en julio de 2011 por JetBrains
- Uno de los objetivos establecidos de Kotlin es el de compilar tan rápido como Java



2000 - Presente

DART

- Fue revelado el 10 de octubre de 2011
- Es un lenguaje de programación de código abierto, desarrollado por Google



3. ERRORES COSTOSOS POR SOFTWARE

CareFusion Alaris (8100)

2015

no retrasaba correctamente una infusión cuando se utilice la opción "Retrasar hasta" o la función "Multidosis"

2020 la compañía retiro de 774.000 bombas Alaris de clase I

55 lesiones y una muerte.



La agencia espacial declara el satélite astronómico como una pérdida.



Informática Año 2000: Una Bomba De Relojería

También conocido como bug Y2K, este error informático fue causado por la costumbre programar la fecha omitiendo el siglo asumiendo que el software sólo estaría en funcionamiento entre 1900 y 1999, y que amenazaba con poner el mundo y dispositivos informáticos en una gran desconfiguración.



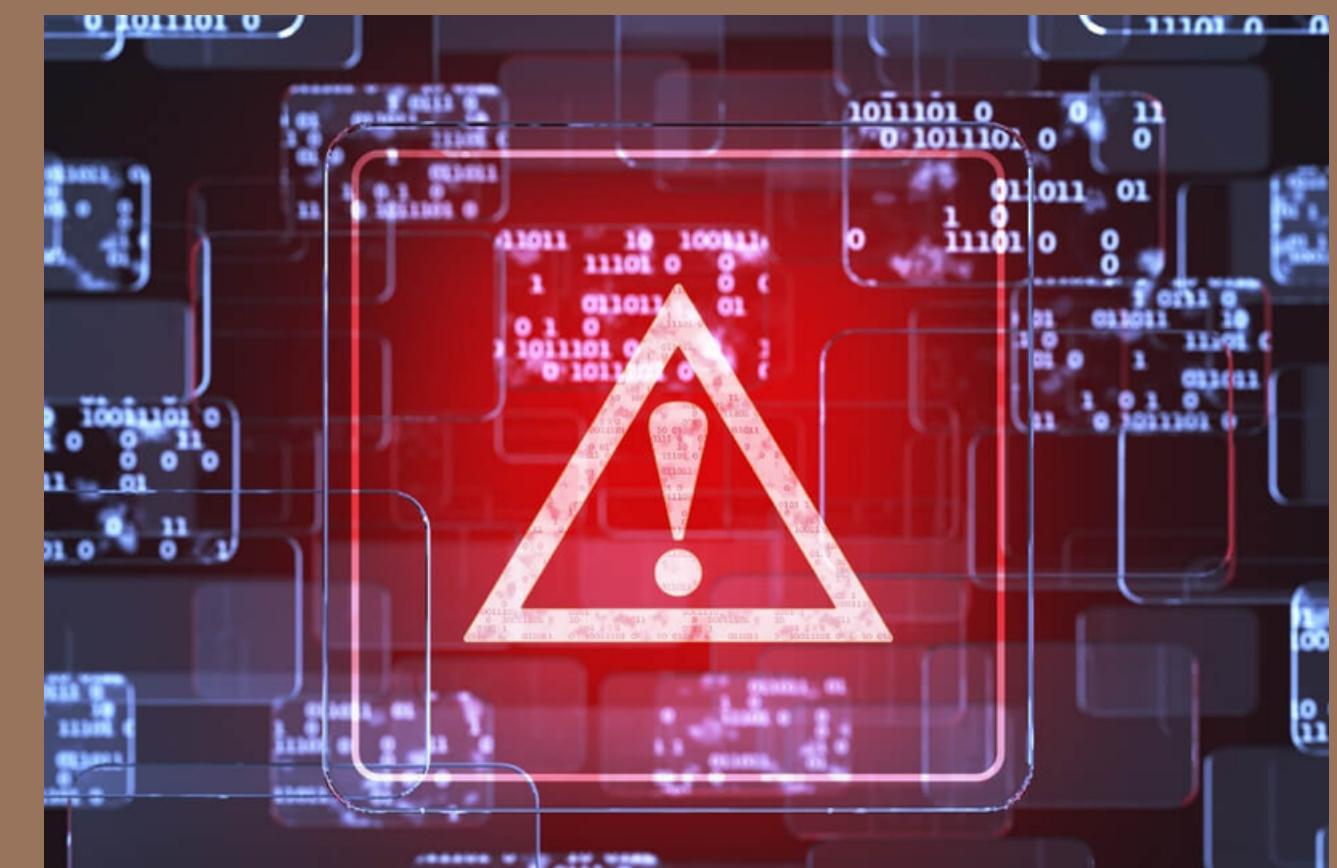
■ Un caso ¿divertido?

- El computador de un hospital comete un error fatal: "De acuerdo con esto, yo estoy muerto"
- <... El problema sucedió durante una actualización rutinaria de los ficheros del ordenador del [hospital] Saint Mary's en octubre, Jennifer Cammenga, la portavoz del Saint Mary's, declaró al Grand Rapid Press...>. <... "Un dígito fue omitido en el código, indicando que los pacientes habían fallecido, en lugar de indicar que habían sido dados de alta"...>



(Noticia de prensa del 8 de enero de 2003)

En mayo de 2001 la Agencia Internacional para la Energía Atómica declaró una emergencia radiológica en Panamá. 28 pacientes sufrieron una exposición, 8 murieron, y $\frac{3}{4}$ partes de los supervivientes pueden sufrir serias complicaciones que en algunos casos pueden llegar a ser mortales. Se concluyó que uno de los factores que provocaron el accidente se debió a un error en el software que controlaba ciertas entradas de datos.



Caso de error costoso por software

En octubre de 2016, Interlogix, un fabricante de dispositivos de pánico personales inalámbricos, retiró del mercado alrededor de 67 000 dispositivos debido a su incapacidad para operar durante situaciones de emergencia.

La causa probable de esta falla en las operaciones fue que el dispositivo no pudo comunicarse con el sistema de seguridad durante un evento de emergencia. La salida era que el fabricante reemplazará los dispositivos.



Dispositivo defectuoso por error de comunicación con el sistema de alerta



4. CALIDAD DE SOFTWARE

ISO 9126

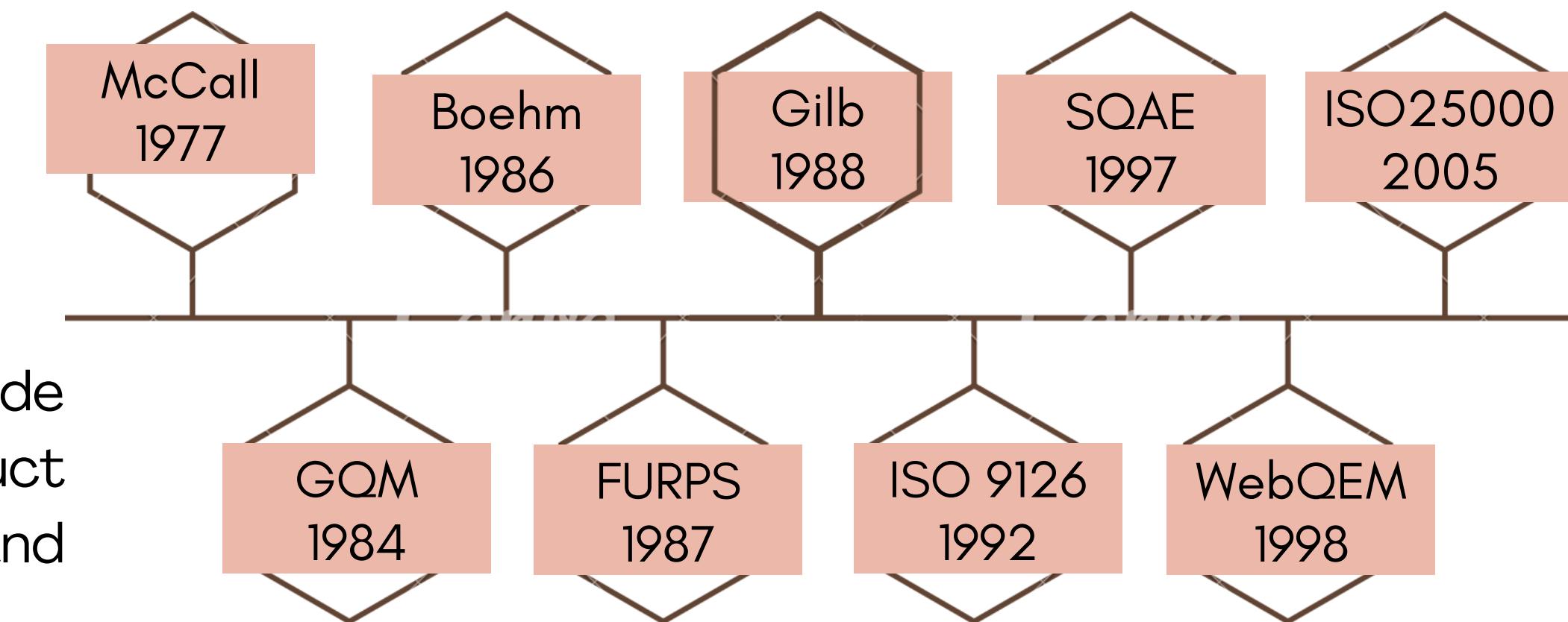
Univ. Mariel Karime Osco Yanapatzí

*ES UN MODELO ESTANDAR
INTERNACIONAL, INVOLUCRADO
EN EL PROCESO DE
CONSTRUCCIÓN DE SOFTWARE*

- **Fue publicado en 1992**, con el nombre de “Information technology – Software product evaluation: Quality characteristics and guidelines for their use”
- **Está basado en el modelo de McCall (1977)**

Línea de tiempo

Evolucion de modelos de calidad a nivel producto



Etapas del desarrollo de software

- Planificación
- Análisis
- Diseño
- Implementación
- Pruebas
- Despliegue
- Uso y mantenimiento

Partes

- Modelo de calidad ISO 9126-1
- Métricas externas ISO 9126-2
- Métricas internas ISO 9126-3
- Calidad de métricas en uso ISO 9126-4



Características

1

Funcionalidad

- Adecuación
- Exactitud
- Interoperabilidad
- Conformidad
- Seguridad

3

Usabilidad

- Comprensibilidad
- Facilidad de aprender
- Operabilidad

2

Confiabilidad

- Nivel de Madurez
- Tolerancia de fallas
- Recuperación

4

Eficiencia

- Comportamiento con respecto al tiempo
- Comportamiento con respecto a recursos

5 Mantenibilidad

- Capacidad de análisis
- Capacitación de modificación
- Estabilidad
- Facilidad de prueba

6 Portabilidad

- Adaptabilidad
- Facilidad de instalación
- Capacidad de remplazo



5. Cuáles serán los requisitos mínimos de una computadora ?, para desarrollar un proyecto de software.

Univ. Edwin Eduardo Gomez Paillo



Desde una etapa de la planificación y análisis, se determina el ámbito al cual va dirigido el proyecto “usuario final”.

Dependiendo del tipo de software que se desarrollara se determinara que requisitos mínimos tendrán las computadoras.



Entidad bancaria



Empresa privada



Usuario general



Requisitos de computadoras de un software para transacciones en entidades financieras



Aquí las computadoras ayudaran realizar transacciones y procesar información de los clientes.

Las computadoras deben tener capacidades de cifrado de hardware y software para evitar que los datos se vean comprometidos durante una transmisión

Cifrado de hardware



Existirá un procesador dedicado donde aloja las funciones matemáticas necesarias para ejecutar el algoritmo de cifrado y descifrado de datos, el cual se almacenara en una memoria aislada del sistema.

Cifrado de Software



El sistema obtiene la información por parte del usuario, estos datos confidenciales son almacenados de forma centralizada y protegidos mediante un cifrado robusto.

La siguiente acción consiste en generar un token único asociado al dato confidencial, incorporándos e al flujo operativo sin revelar ninguna información

Computadora central (Servidor)

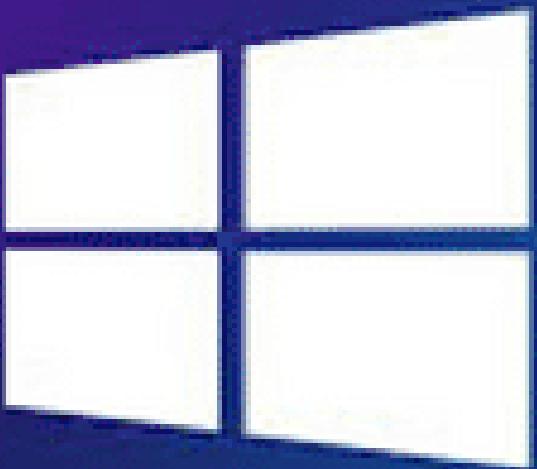
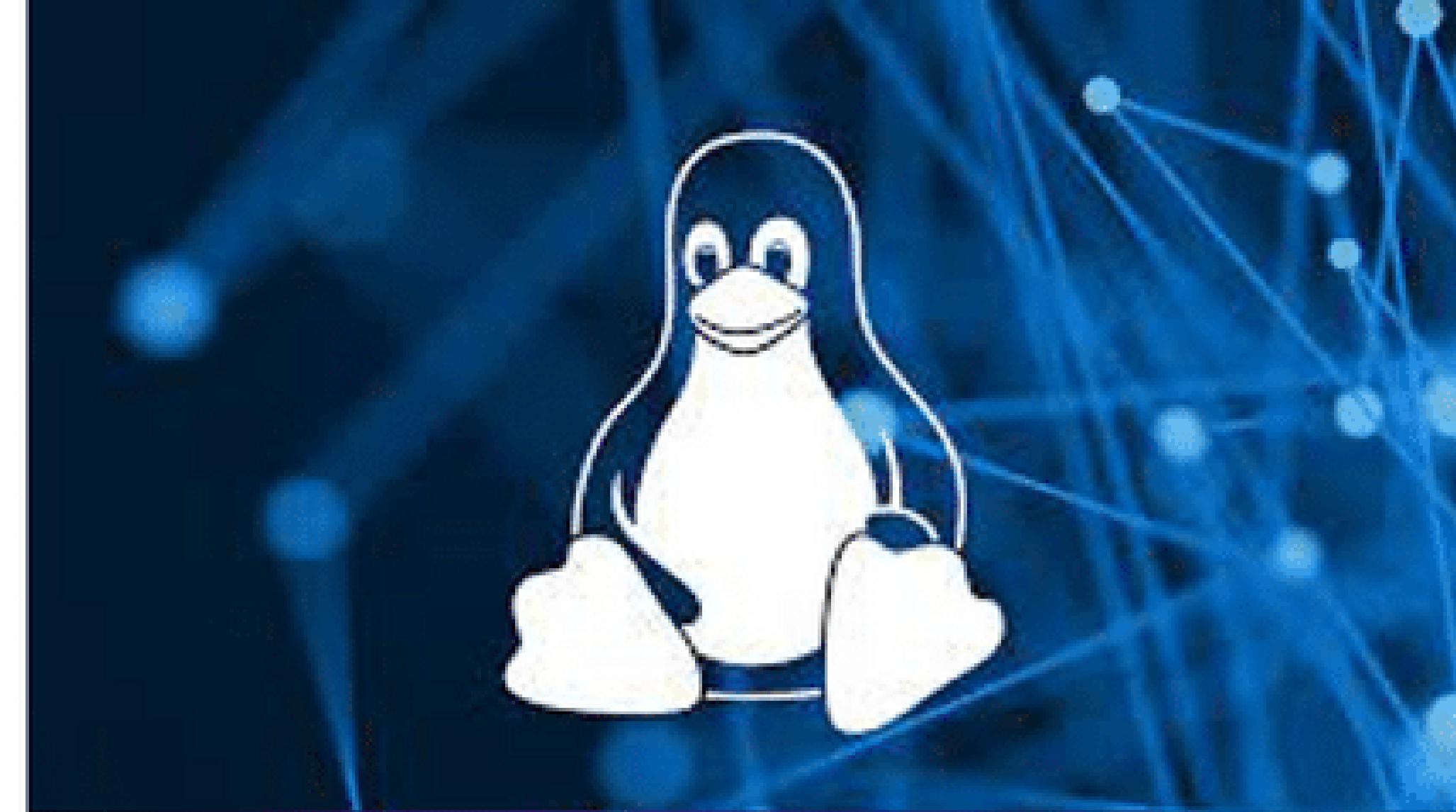
Estas computadoras deben realizar varias instrucciones por segundo tienen características muy específicas como trabajar de manera ininterrumpida ,estructura centralizada , diseño en entradas y salidas de gran volumen.



Los requerimientos de Sistemas Operativos

En los servidores se usa un sistema operativo mas usado es Linux que es un servidor estable y fiable, y tienes múltiples medidas que se pueden implementar en estas.

Desde su propio firewall IP tables, hasta TCP wrappers y las medidas de seguridad SELinux



Computadora central (Servidor)

Estas computadoras deben realizar varias instrucciones por segundo tienen características muy específicas como trabajar de manera ininterrumpida ,estructura centralizada , diseño en entradas y salidas de gran volumen.



Computadoras de apoyo

Estas deben estar programadas para ejecutar software de hojas de cálculo, procesamiento de textos y bases de datos, además de navegadores web y clientes de correo electrónico.

Deben estar conectadas mediante una red área local de área local (LAN), para su comunicación entre ellas.



De manera general Windows
estará instalado
computadoras de apoyo,
desempeñando funciones
calculo y registro de datos.



**Requisitos para
computadores
que usen un
software de
paginas webs**



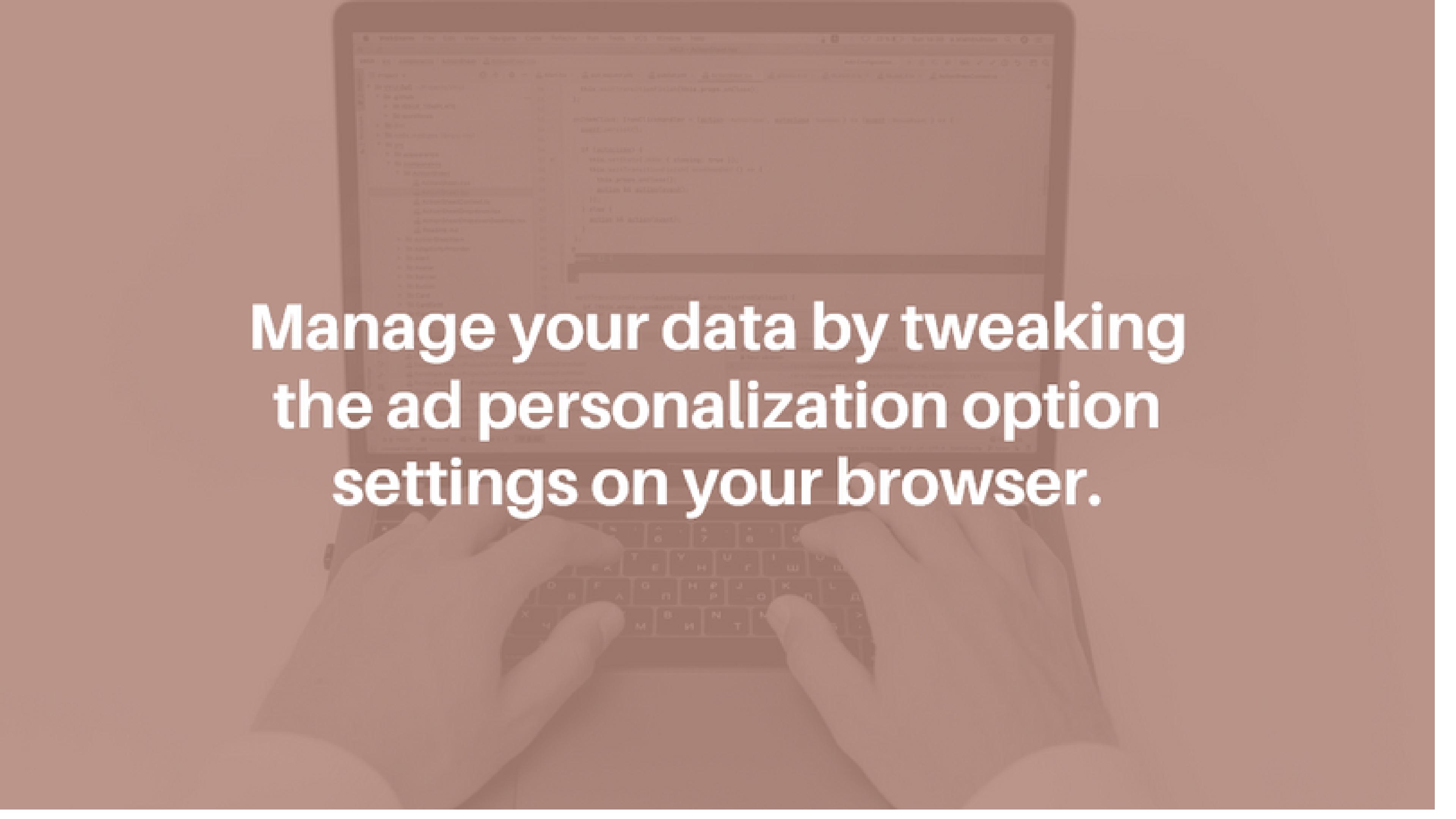
No requiere ordenadores potentes, pues solo utilizan los ordenadores para navegar en ese tipo de software en la red una computadora de gama baja y media seria suficiente .

- Incrementar la memoria RAM, para usar simultáneamente varias pestañas y trabajar en el ordenador .
- Respecto al S.O., requerido en la computadora Windows por ser el mas usado para acceder a red seguido de Android.



Requisitos para computadoras para Software de juegos





Manage your data by tweaking
the ad personalization option
settings on your browser.

Los requisitos son específicos y exigentes en cuanto el software y hardware .

La tarjeta de video (GPU) es la mas importante, ya que un juego para su fluides y buen rendimiento del mismo en cuanto a los FPS del computador.

Se requiere al menos una tarjeta de video NVIDIA GeForce GTX 1050 o AMD RX 560 en paralelo con una cantidad de memoria de video igual o superior a 4 GB.

Así también una memoria RAM mas alta para el buen rendimiento de la PC.



**GEFORCE
RTX**



Los requisitos en cuanto el software.

Sistema Operativo Windows por ser el más popular la mayoría de los juegos se basan en la API de DirectX, que está desarrollada por Microsoft y está disponible solo en Windows.

Linux no es tanto, ya que hay muy pocos juegos compatibles con este sistema operativo.

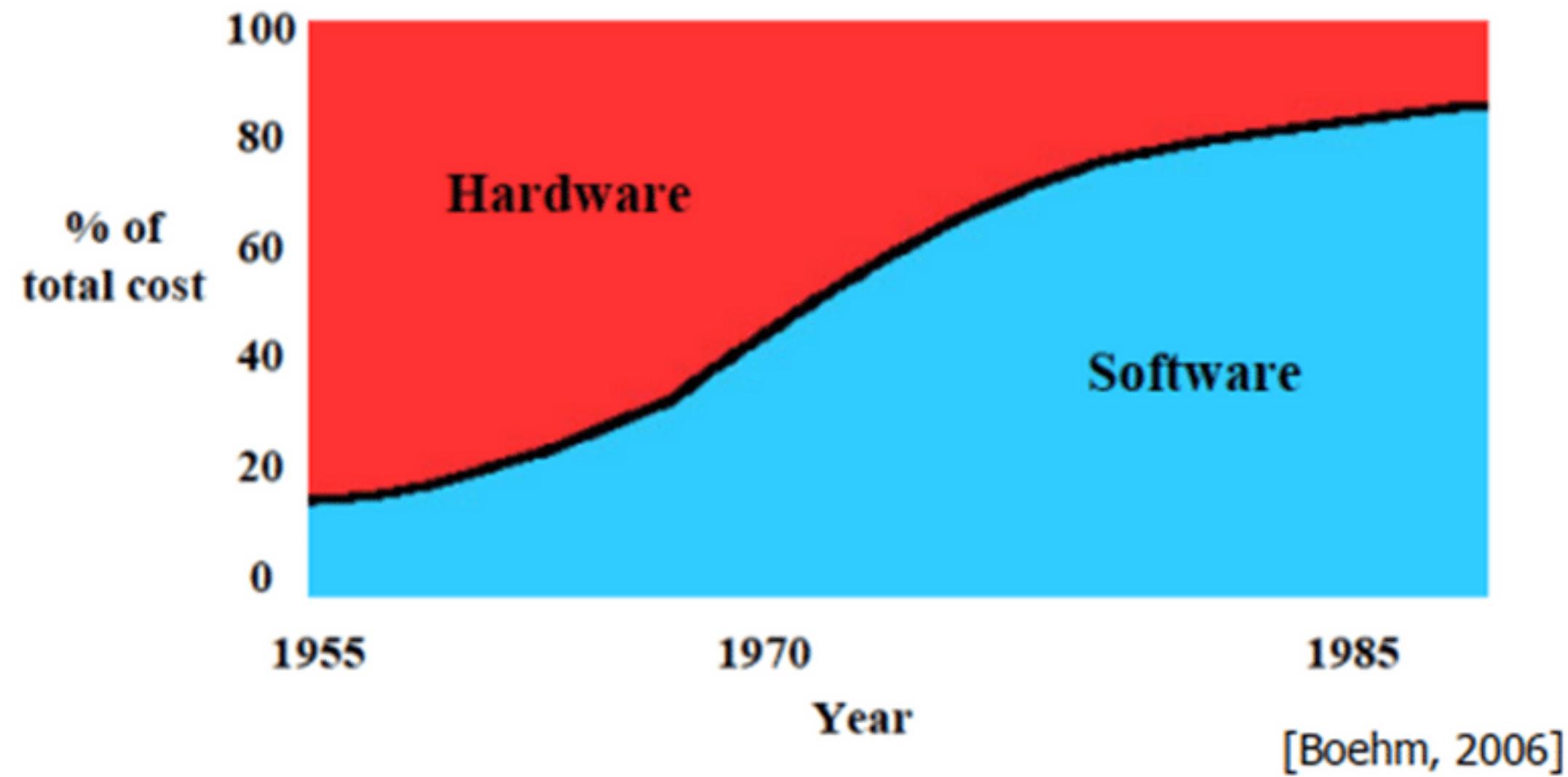
También se pueden desarrollar juegos para macOS que es de Apple.



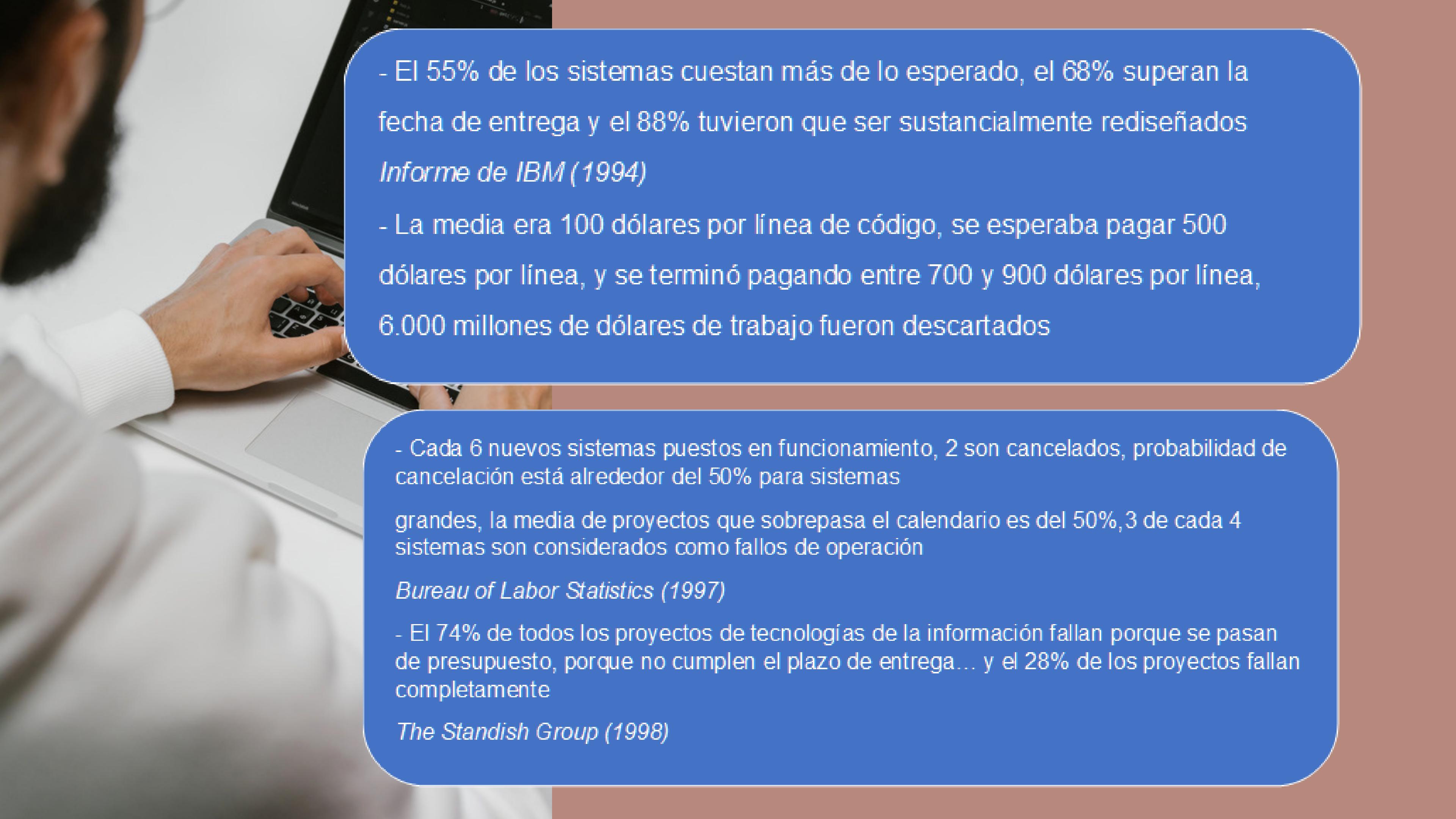
6.Como se calcula el costo de un software

Univ. Ivan Israel Gutierrez

- Evolución de los costes del *software*



El software es un producto de consumo con un gran peso en la economía mundial principalmente en los países desarrollados

- 
- El 55% de los sistemas cuestan más de lo esperado, el 68% superan la fecha de entrega y el 88% tuvieron que ser sustancialmente rediseñados

Informe de IBM (1994)

- La media era 100 dólares por línea de código, se esperaba pagar 500 dólares por línea, y se terminó pagando entre 700 y 900 dólares por línea, 6.000 millones de dólares de trabajo fueron descartados

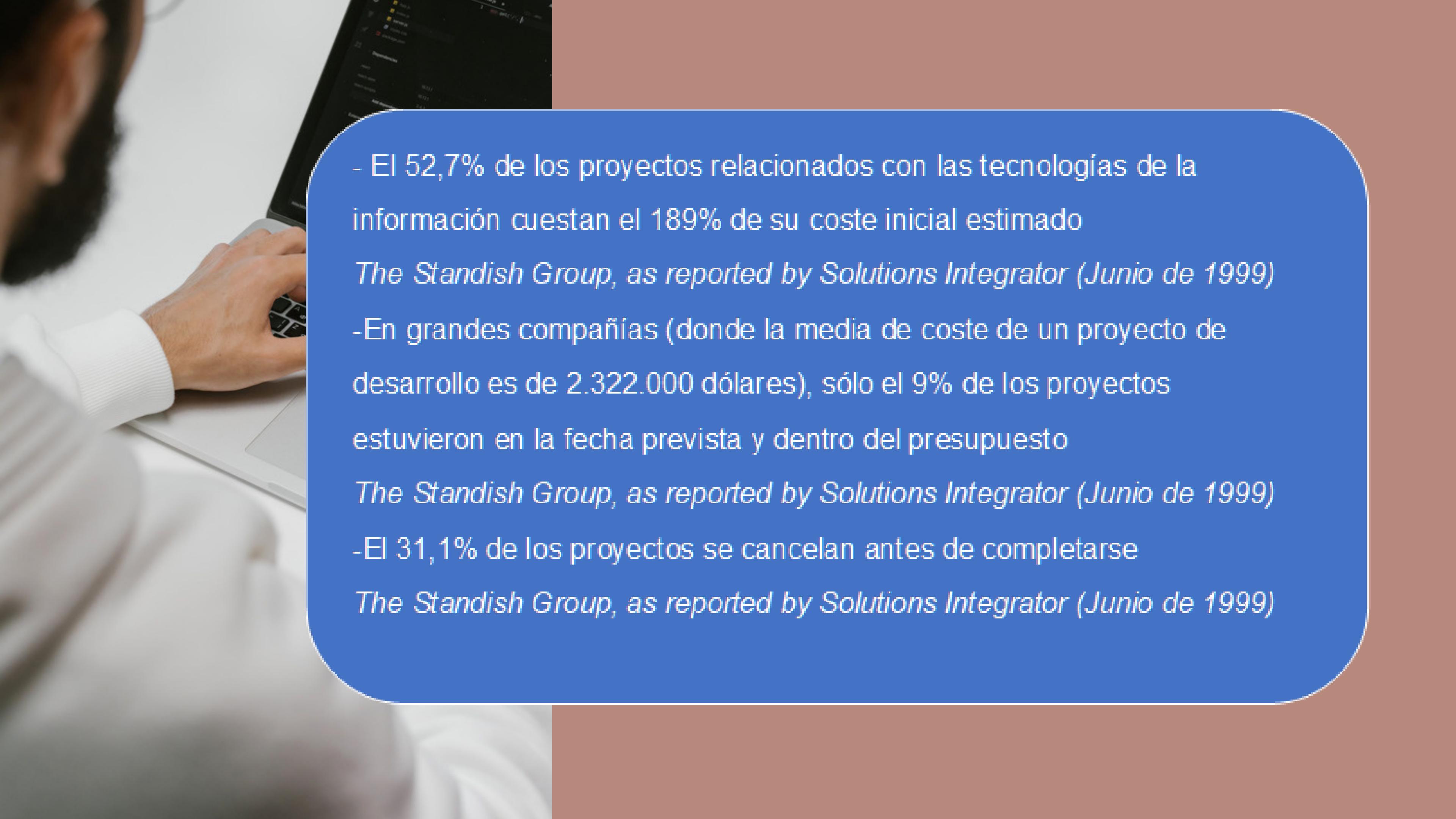
- Cada 6 nuevos sistemas puestos en funcionamiento, 2 son cancelados, probabilidad de cancelación está alrededor del 50% para sistemas

grandes, la media de proyectos que sobrepasa el calendario es del 50%, 3 de cada 4 sistemas son considerados como fallos de operación

Bureau of Labor Statistics (1997)

- El 74% de todos los proyectos de tecnologías de la información fallan porque se pasan de presupuesto, porque no cumplen el plazo de entrega... y el 28% de los proyectos fallan completamente

The Standish Group (1998)

- 
- El 52,7% de los proyectos relacionados con las tecnologías de la información cuestan el 189% de su coste inicial estimado

The Standish Group, as reported by Solutions Integrator (Junio de 1999)

- En grandes compañías (donde la media de coste de un proyecto de desarrollo es de 2.322.000 dólares), sólo el 9% de los proyectos estuvieron en la fecha prevista y dentro del presupuesto

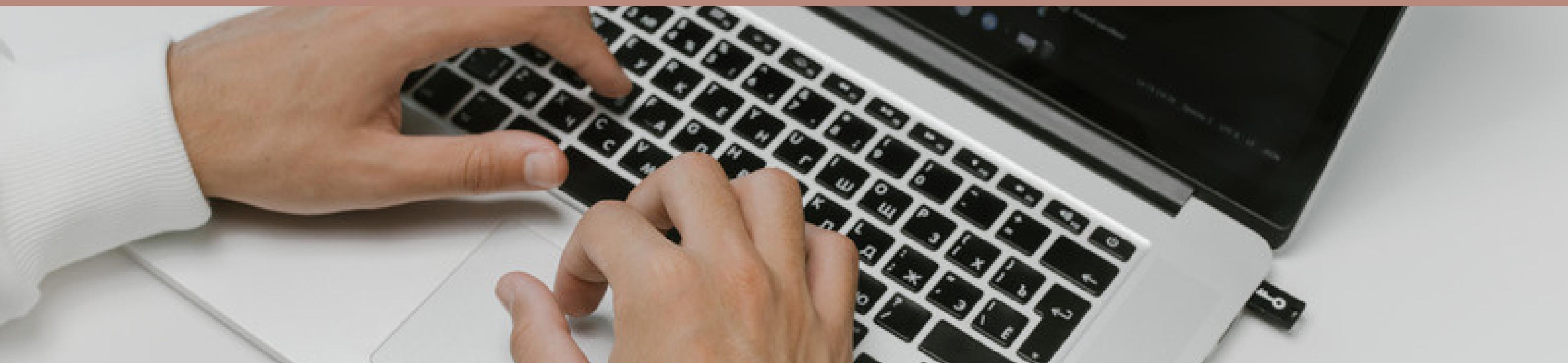
The Standish Group, as reported by Solutions Integrator (Junio de 1999)

- El 31,1% de los proyectos se cancelan antes de completarse

The Standish Group, as reported by Solutions Integrator (Junio de 1999)

Estimación de costos.

Análisis Costo-Beneficio:



La técnica del análisis costo/beneficio tiene como objetivo fundamental proporcionar una medida de los costos en que se incurre en la realización de un proyecto y comparar dicha previsión de costos con los beneficios esperados de la realización de dicho proyecto. A la hora de realizar el cálculo de los costos se deben considerar, entre otros, elementos como los siguientes: (MAP, 2001)

Adquisición y mantenimiento de hardware y software.

Gastos de comunicaciones (líneas, teléfono, correo, etc.).

Gastos de instalación (cableado, acondicionamiento de sala, recursos humanos y materiales, gastos de viaje, etc.).

Costo de desarrollo del sistema.

Gastos (costo anual) del mantenimiento del sistema

Gastos de consultoría: En caso de requerirse algún consultor externo en cualquier etapa del proyecto.

Gastos de formación: de todo tipo de personal (desarrolladores, operadores, implantadores, usuarios finales, etc.).

Gastos de material: Papel, toner, etc.

Costos derivados de la curva de aprendizaje del personal involucrado Costos financieros, de publicidad, etc.

Y para la estimación de beneficios se deben considerar cuestiones como las siguientes:

Incremento de la productividad: Ahorro o mejor utilización de recursos humanos.
Ahorro de gastos de mantenimiento del sistema actual.

Ahorros de adquisición y mantenimiento de hardware y software, o reutilización de plataformas sustituidas.

Incremento de ventas o resultados, y disminución de costos producidos por una mejora de la gestión (rotación de stock, “just in time”, gestión de relaciones con clientes, etc.).

Ahorro de material de todo tipo: Sustituido por datos electrónicos que proporciona el sistema, como por ejemplo: papel, correo, etc.

Calculo de costos del desarrollo software

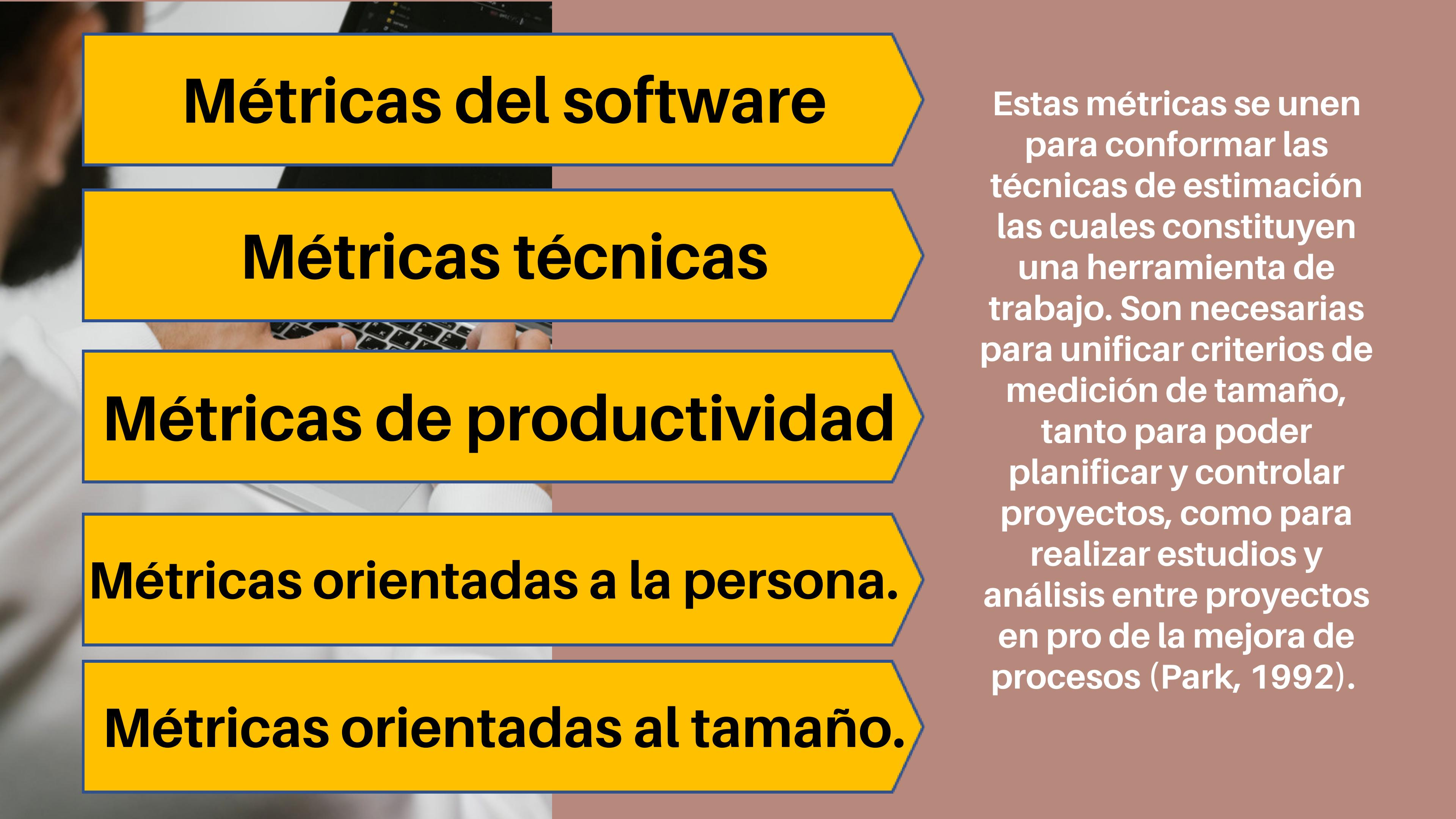




El proceso de estimación del costo de un producto software está formado por un conjunto de técnicas y procedimientos que se usan en la organización para poder llegar a una predicción fiable. Este es un proceso continuo, que debe ser usado y consultado a lo largo de todo el ciclo de vida del proyecto. Se divide en los siguientes pasos

Técnicas de estimación de costos en el desarrollo de software





Métricas del software

Métricas técnicas

Métricas de productividad

Métricas orientadas a la persona.

Métricas orientadas al tamaño.

Estas métricas se unen para conformar las técnicas de estimación las cuales constituyen una herramienta de trabajo. Son necesarias para unificar criterios de medición de tamaño, tanto para poder planificar y controlar proyectos, como para realizar estudios y análisis entre proyectos en pro de la mejora de procesos (Park, 1992).

Existen formas de realizar la estimación de los costos éstas pueden ser mediante:

COCOMO

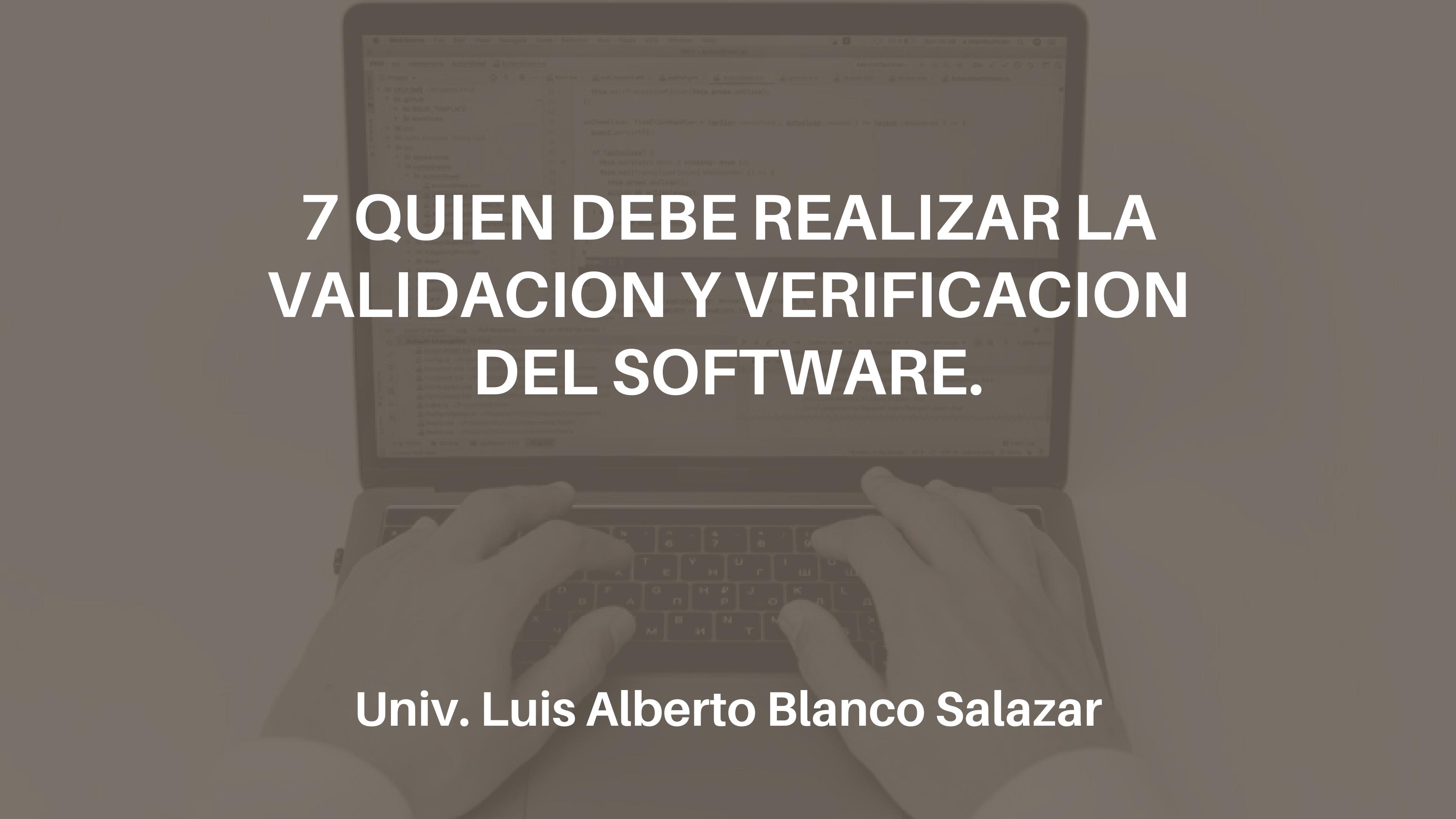
El Modelo COCOMO Creado por Barry Boehm en 1981. Su nombre significa COnstructiveCOst MOdel (Modelo constructivo de costo) constituye una jerarquía de modelos de estimación para el software.

COCOMO II

COCOMO II es un modelo que permite estimar el costo, el esfuerzo y el tiempo cuando se planifica una nueva actividad de desarrollo software, y está asociado a los ciclos de vida modernos.

Técnica Delphi: Permite sistematizar y mejorar la opinión de los expertos consultados.
Analogía: Enfoque más formal que el de la opinión de expertos. Los expertos comparan el proyecto propuesto con uno o más proyectos anteriores intentando encontrar similitudes y diferencias particulares.





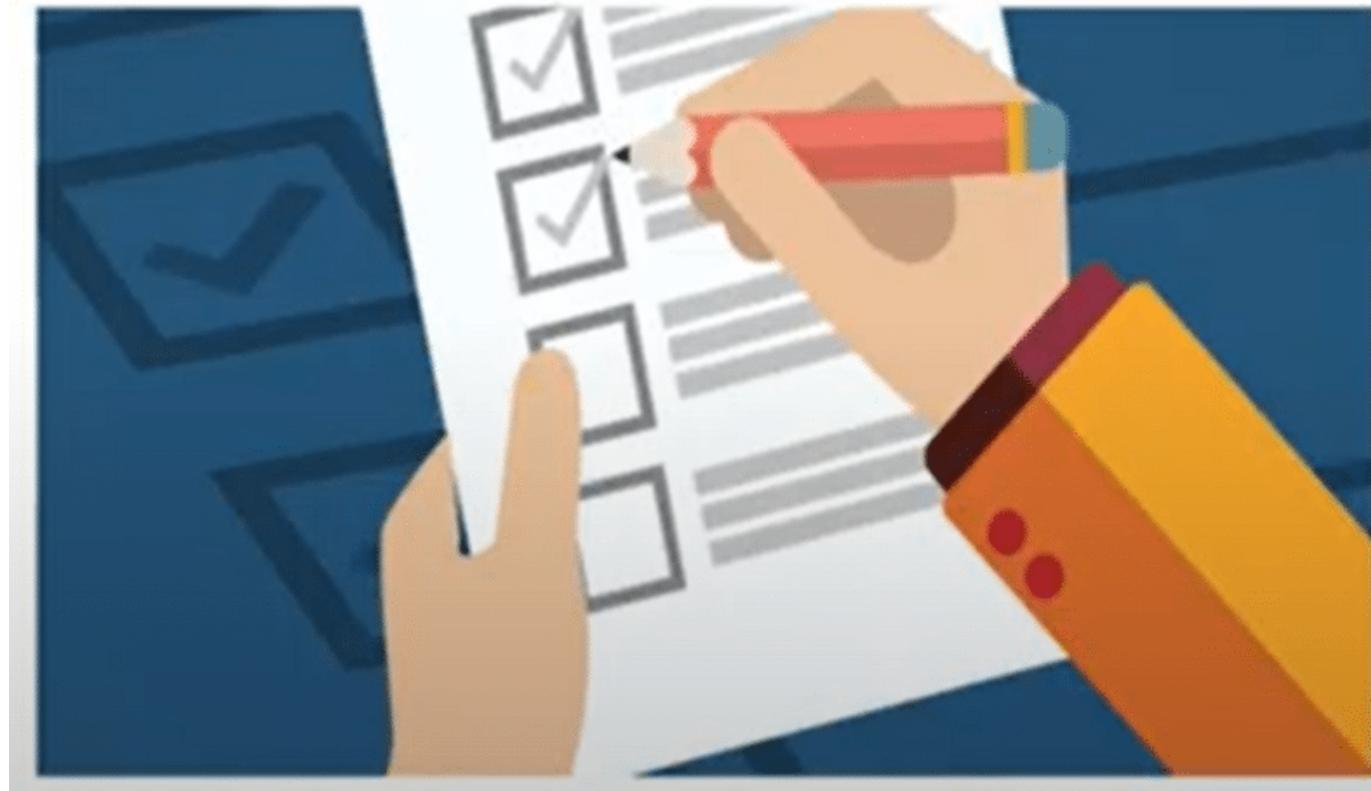
7 QUIEN DEBE REALIZAR LA VALIDACION Y VERIFICACION DEL SOFTWARE.

Univ. Luis Alberto Blanco Salazar

¿Que es la Verificacion y validacion de software?

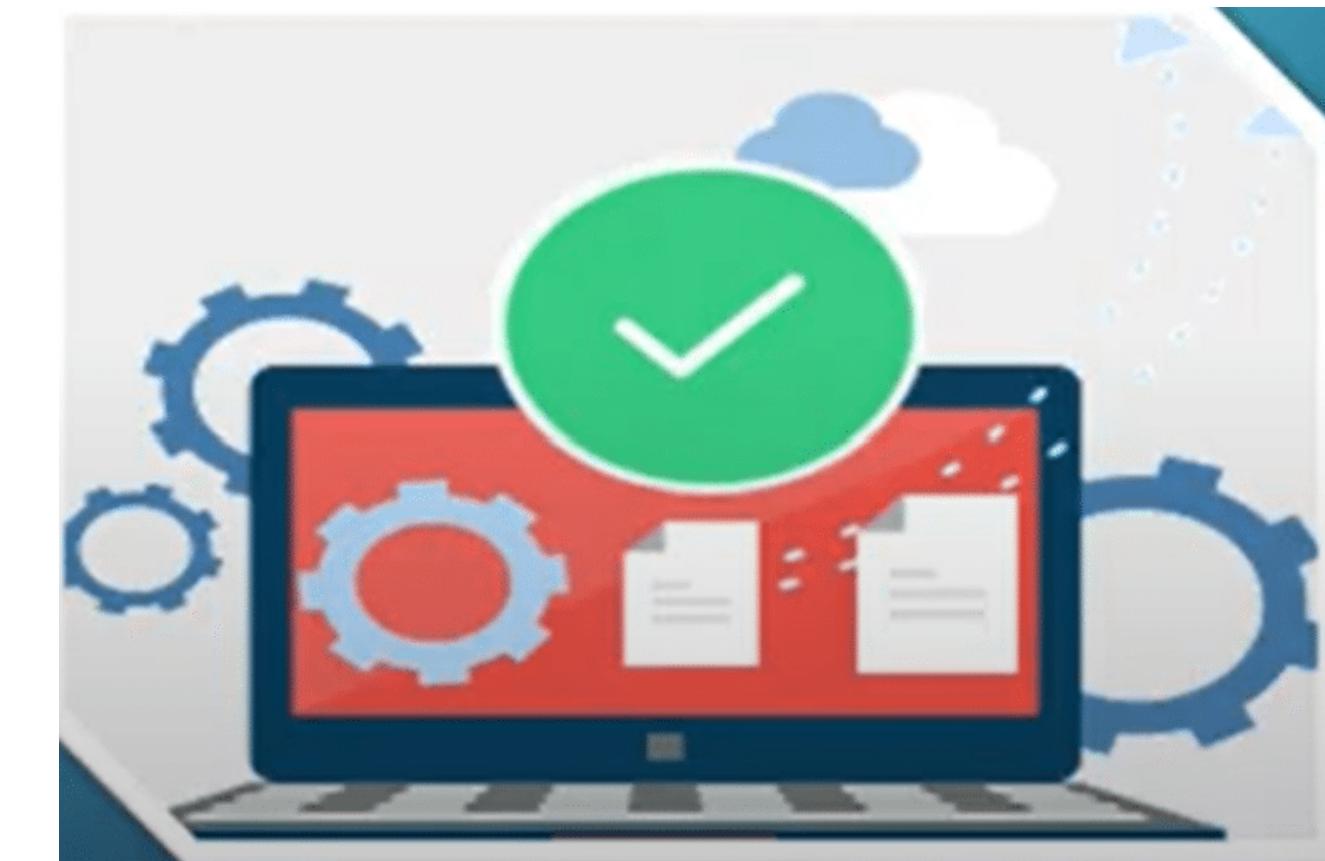
VERIFICACION

¿Estamos construyendo el producto correctamente?



VALIDACION

¿Estamos construyendo el producto correcto?



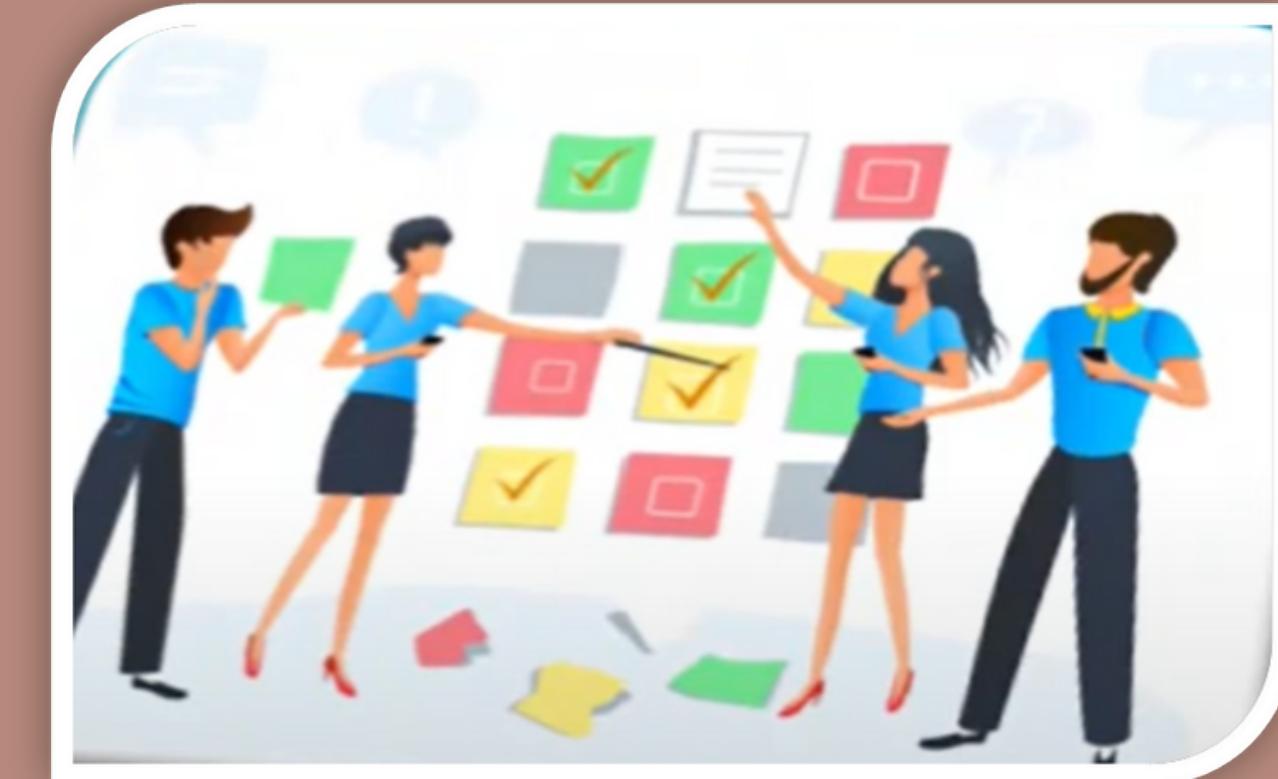
Técnicas de verificación y validación

Inspecciones de software



Las inspecciones de software analizan y comprueban las representaciones del sistema tales como: el documento de requerimientos, los diagramas de diseño y el código fuente del programa.

Pruebas de software



Las pruebas en el proceso de desarrollo implican ejecutar los procesos que realiza el programa desarrollado con el fin de evaluar los diferentes estados que adquiere el sistema

Pruebas de software

Test	Objetivo
Prueba unitario	Detectar errores en los datos, lógica algoritmos
Prueba integración	Detectar errores de interfaces y relaciones entre componentes
Prueba regresión	Comprueba que los cambios efectuados, afecten otras partes
Pruebas de Humo	Comprueba los errores tempranos y de manera fácil.
Pruebas de sistema	Detectar fallas en el cubrimiento de los requerimientos

Pruebas alpha y beta

Prueba alpha



La prueba alfa se lleva a cabo en el sitio del desarrollador por un grupo representativo de usuarios finales. El software se usa en un escenario natural con el desarrollador observando a los usuarios y registrando los errores y problemas de uso.

Prueba beta



Por lo general el desarrollador no está presente. La prueba beta es una aplicación “en vivo” del software en un ambiente que no puede controlar el desarrollador. El cliente registra todos los problemas (reales o imaginarios) que se encuentran durante la prueba beta y los reporta al desarrollador periódicamente.

Validación y verificación de pruebas como solución al mejoramiento de la calidad del software

Los procesos de V&V proporcionan una evaluación objetiva de los productos y procesos en los CVDS. Esta evaluación objetiva demuestra si los requisitos son correctos, completos, precisos, consistentes y verificables

¿Cómo determinar si el sistema está cumpliendo a cabalidad con las necesidades del usuario?

IEEE Std 1012TM-2012

ISO/IEC 15288:2008 - IEEE/EIA

Std12207.0:1996

ISO/IEC 15288:2008

ISO/IEC 12207:2008

La IEEE Std 1012TM-2012

- a) Apoyo en adquisición V&V.
- b) Planificación de suministros V&V.
- c) Planificación del proyecto V&V.
- d) Administración de la configuración V&V.
- e) Definición de requerimientos del sistema (stakeholder) V&V.
- f) Sistema de análisis de requisitos V&V.
- g) Diseño de la arquitectura del sistema V&V, concepto de software V&V, concepto de hardware V&V.
- h) Implementación del sistema V&V, Actividades V&V de hardware y software.



a) JUEGO DE RUGBY - ORIGEN



Todo comienza en 1823, cuando la leyenda William cogió una pelota en medio partido de fútbol y corrió hacia la línea de gol, creando un nuevo deporte.

William Webb Ellis 1806 - 1872

b) Objetivo

El objetivo del juego es marcar la mayor cantidad de puntos posible contra un equipo oponente portando, pasando, pateando y apoyando la pelota, de la linea de goal de acuerdo alas leyes del juego, a su espíritu deportivo y al juego limpio.

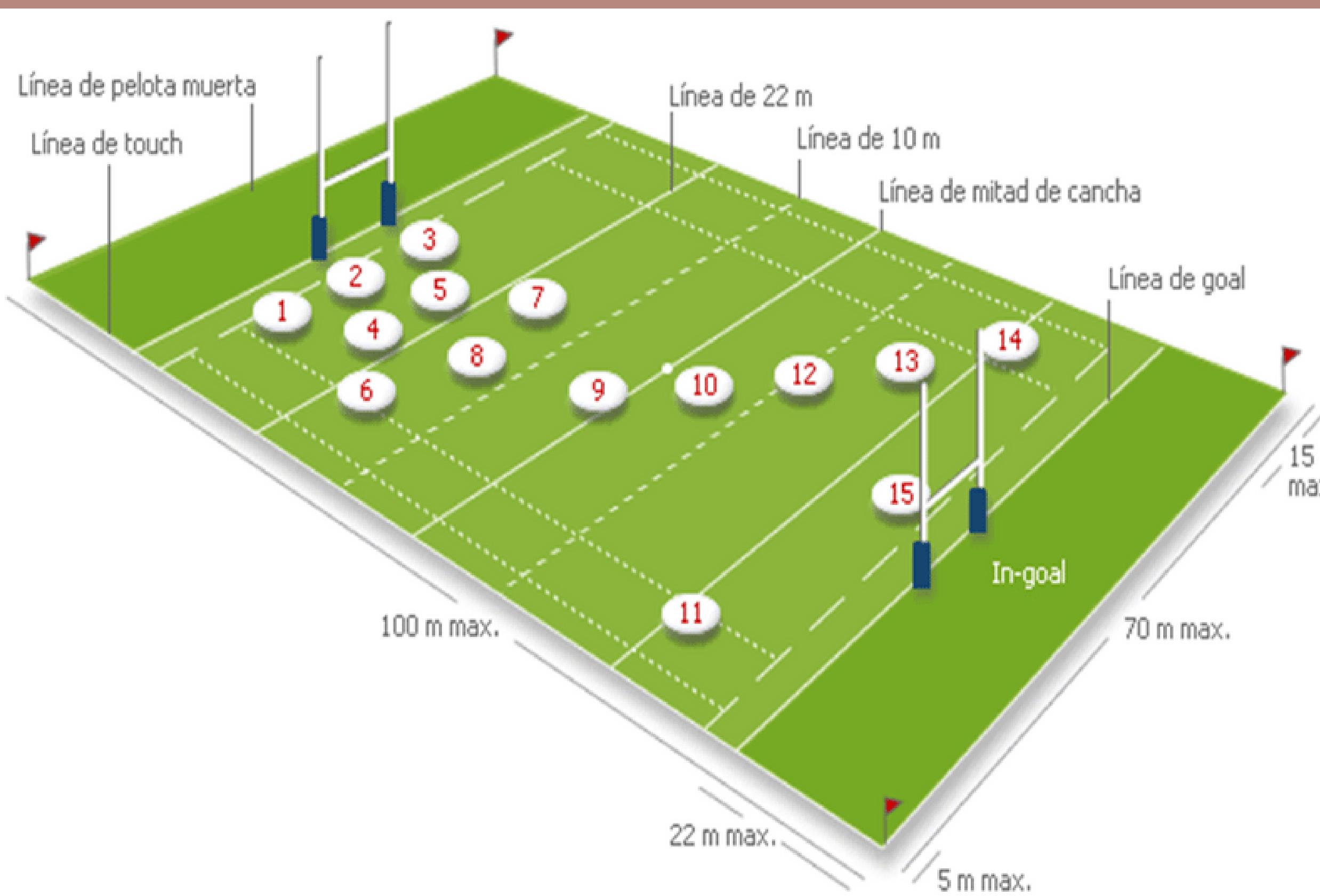


b) Objetivo

El objetivo del juego es marcar la mayor cantidad de puntos posible contra un equipo oponente portando, pasando, pateando y apoyando la pelota, de la linea de goal de acuerdo alas leyes del juego, a su espíritu deportivo y al juego limpio.



Composition



- 1 Pilier (izq - der)
- 2 Talonador
- 3 Segunda Linea (izq - der)
- 4 Flanker (izq - der)
- 5 Tercera centro
- 6 Medio Melé
- 7 Medio Apertura
- 8 Ala Izquierdo
- 9 Primer Centro
- 10 Segundo Centro
- 11 Ala derecho
- 12 Zaguero

d) Reglas del juego

- Los partidos de rugby se dividen en dos tiempos de 40 minutos cada uno.
- El balón sólo puede avanzar llevándolo o pateándolo hacia adelante.
- El ganador del volado puede escoger una de dos opciones. Puede escoger en qué dirección atacará o puede escoger hacer la salida de mitad de campo
- Un jugador placado o tacleado (derribado) debe pasar o soltar inmediatamente el balón. El jugador que taclea debe también soltar inmediatamente al jugador tacleado.
- El rugby es un deporte continuo. No se prevé interrupciones (a menos que haya una lesión.).
- Una mele/scrum reinicia el juego después de un pase hacia adelante o un knock-on. También se forma una mele/ scrum en otras ocasiones menos frecuentes.



d) Reglas del juego

- **Un line-out reinicia el juego cuando el balón sale del terreno de juego.**
- **Un Ensayo/Try es otorgado cuando el balón es llevado más allá la línea de goal (zona de anotación) y apoyado en el suelo.**
- **5 puntos se otorgan al realizar un ensayo/try.**
- **2 puntos se otorgan al convertir la patada adicional después de un ensayo/try.**
- **3 puntos se otorgan al convertir un gol de campo (golpe) después de cometida una infracción.**
- **3 puntos se otorgan al convertir un drop (patada de bote-pronto) en juego abierto.**



d) Reglas del juego

- Despu s de que se convierte un ensayo/try o un penal, el bal n es pateado hacia el equipo anotador (excepto en sevens, rugby con siete jugadores por lado).
- El  rbitro es el responsable de hacer respetar el reglamento.
- Se juega en dos tiempos continuos de 40 minutos cada uno con un intermedio de 5 minutos.
- El tiempo lo lleva el  rbitro principal y debe detenerlo solamente cuando haya lesiones.
- Hay dos jueces de l nea que ayudan a indicar cu ndo el bal n o la persona que lo lleva salen del campo de juego.



d) Reglas del juego

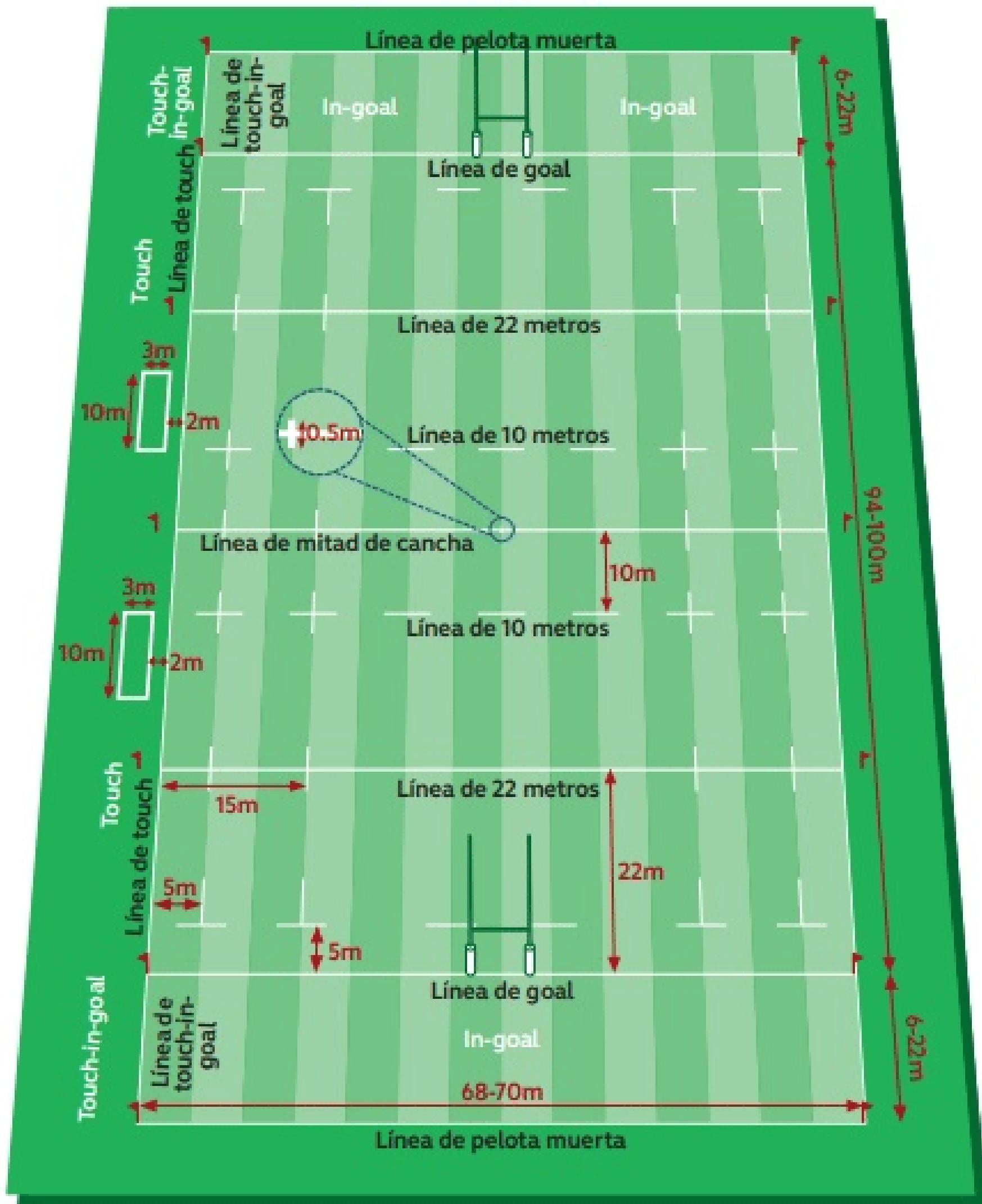
- Un line-out reinicia el juego cuando el balón sale del terreno de juego.
- Un Ensayo/Try es otorgado cuando el balón es llevado más allá la línea de goal (zona de anotación) y apoyado en el suelo.
- 5 puntos se otorgan al realizar un ensayo/try.
- 2 puntos se otorgan al convertir la patada adicional después de un ensayo/try.
- 3 puntos se otorgan al convertir un gol de campo (golpe) después de cometida una infracción.
- 3 puntos se otorgan al convertir un drop (patada de bote-pronto) en juego abierto.



d) Reglas del juego

El campo de juego debe ser rectangular, con un largo de 100 metros por 70 metros de ancho.

Las superficies permitidas para el terreno son césped, arena, tierra, nieve y césped artificial



e) Valores notables que deben cumplir cada uno de los jugadores

Del 1 al 8 todos son delanteros o forwards, los encargados de conseguir las anotaciones.

Del 9 al 15 se les llama en general “backs” y son los que se colocan en la línea de tres cuartos, por lo general se ocupan de defender.

