

Control de Calidad



SDLC

¿QA sólo está en las pruebas?

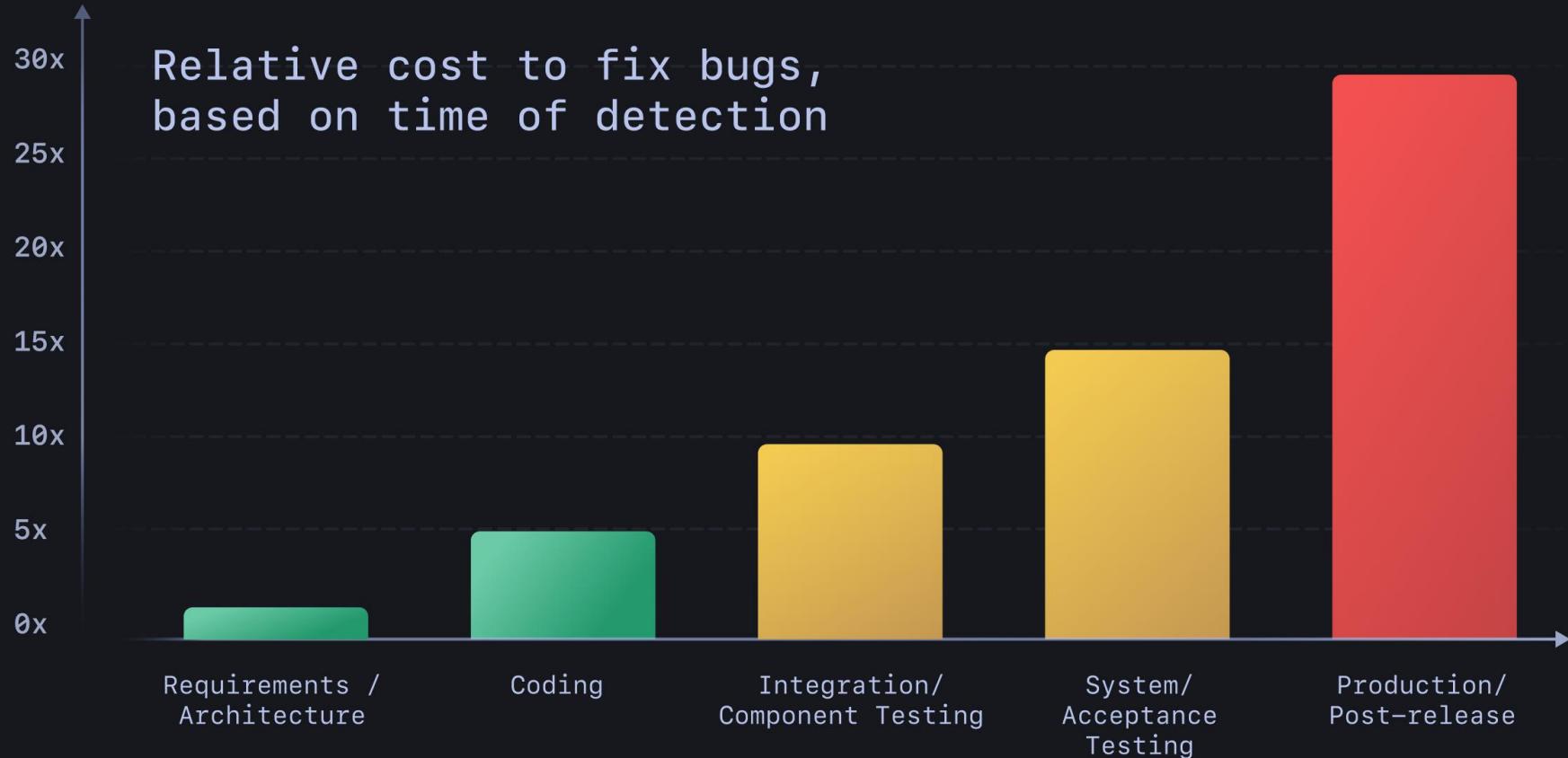
1

- ◊ Análisis de requerimientos → ◊ Comprendión de requerimientos
- ◊ Diseño → ◊ Crear escenarios de pruebas
- ◊ Desarrollo → ◊ Pruebas unitarias
- ◊ Pruebas → ◊ Funcionales, integración, seguridad, etc.
- ◊ Implementación → ◊ Pruebas del cliente(UAT), pruebas en entorno de prod.
- ◊ Mantenimiento → ◊ Replicar posibles errores

QA

¿Por qué es importante?





Validación vs Verificación

- ◊ Verificación: que el sistema de información funcione correctamente como fue diseñado.
- ◊ Validación: que el sistema de información cumpla con los requerimientos funcionales.

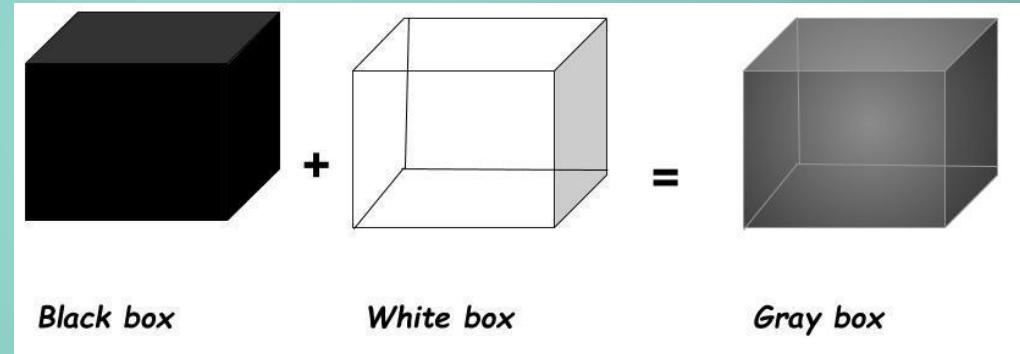
QA

Tipos de pruebas

3

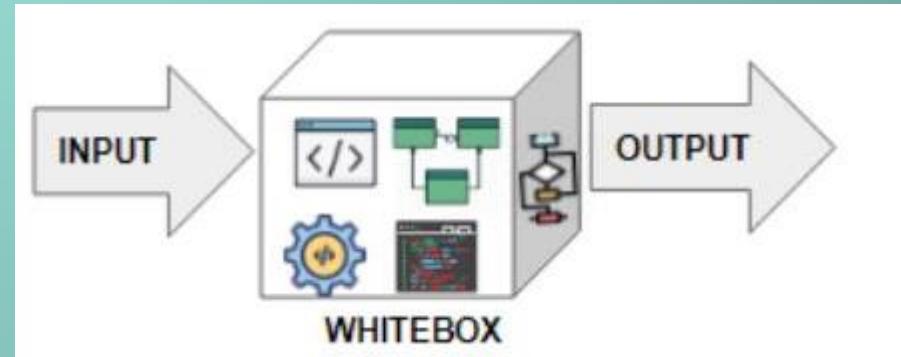
Pruebas Funcionales

- ◊ Caja Blanca
- ◊ Caja Negra
- ◊ Caja Gris



Pruebas Caja blanca

- ◊ Pruebas unitarias
- ◊ Algunas pruebas de seguridad
- ◊ Verificación de calidad de código con ciertas herramientas



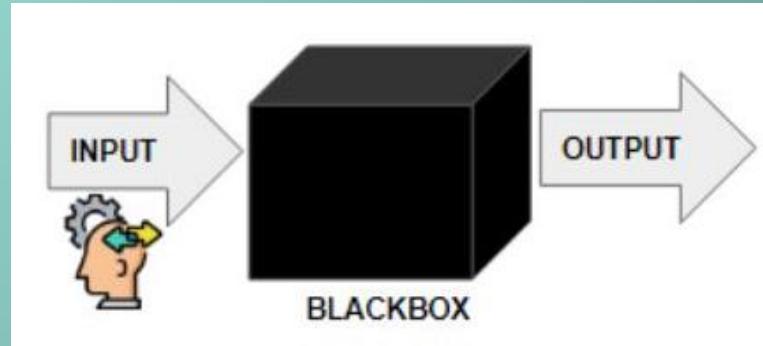
Pruebas Caja Gris

- ◊ Pruebas funcionales de backend
- ◊ Algunas pruebas funcionales de front end
- ◊ Pruebas de integración
- ◊ Pruebas de carga
- ◊ Pruebas de estrés
- ◊ Scanners de seguridad



Pruebas caja Negra

- ◊ Pruebas funcionales de front end(sin acceso a logs u otras herramientas)
- ◊ User Acceptance Testing
- ◊ Pruebas de experiencia de usuario
- ◊ Alfa
- ◊ Beta



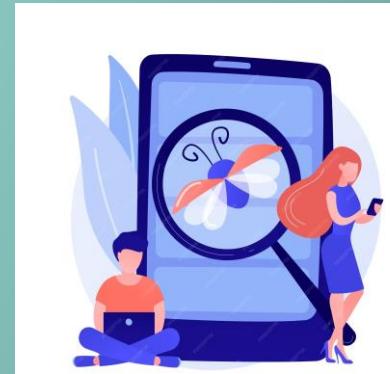
Pruebas Alfa

- ◊ Pruebas en presencia del equipo de desarrollo
- ◊ Pruebas por parte de algún representante del cliente
- ◊ Monitoreo constante
- ◊ El sistema no está del todo terminado



Pruebas Beta

- ◊ Grupo de los usuarios o clientes finales
- ◊ El desarrollador no está presente
- ◊ Se crean logs muy detallados



Pruebas de aceptación de usuario

- ◊ La aplicación está terminada pero antes de pasar a producción...
- ◊ El cliente inicia pruebas propias con data real
- ◊ Asegurar que información de producción funcione sea compatible



Pruebas Funcionales

- ◊ Pruebas que tienen como objetivo probar tanto los escenarios esperados como negativos basados en los casos de uso



Las pruebas funcionales pueden:

- ◊ Probar la interfaz gráfica
- ◊ Probar backend

FRONT END

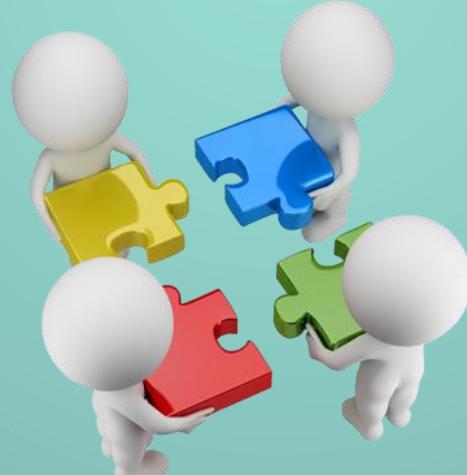


BACK END



Pruebas de integración

- ◊ Verificar que diferentes servicios o módulos se acoplen correctamente
- ◊ Los tipos de datos deben coincidir



Pruebas de experiencia de usuario

- ◊ Verificar qué tan intuitivo es el sistema
 - Controles fáciles de usar
 - Distribución lógica de elementos en pantalla
 - Instrucciones comprensibles



QA

Pruebas automáticas



Pruebas Unitarias

- ◊ Pruebas sobre métodos y funciones
- ◊ A nivel de código
- ◊ **Code Coverage**

xUnit.net



TestNG

JUnit

Backend automático

- ◊ Simular llamadas del sistema
- ◊ Asserts pero a nivel de respuestas



REST-Assured

GET, POST, PUT, PATCH, DELETE



SoapUI



SMARTBEAR
ReadyAPI



UI Automática

- ◊ Simular las acciones de un usuario



Playwright



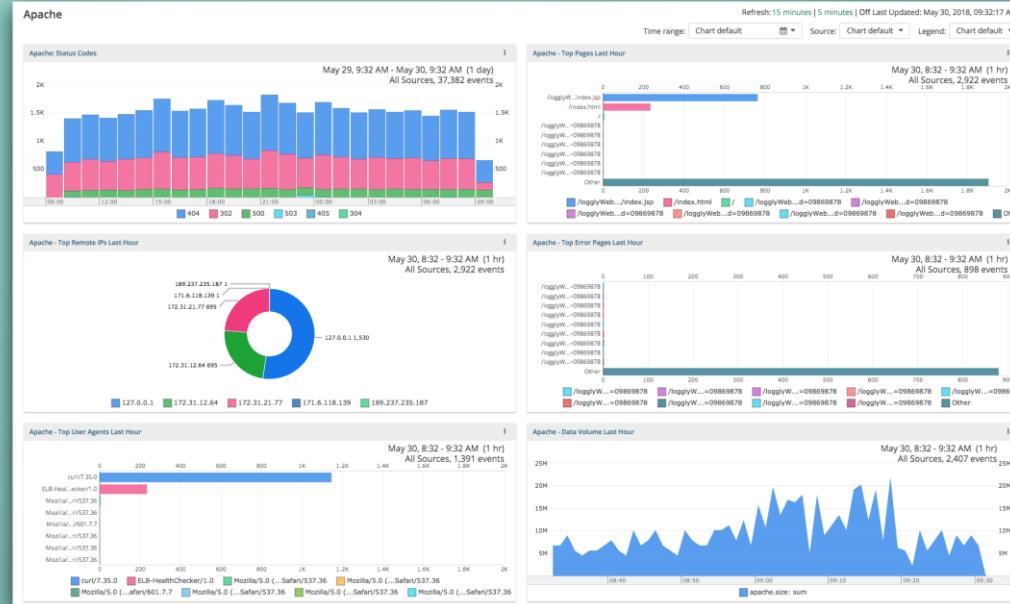
Pruebas de carga

- ◊ Asegurar la cantidad de transacciones soportadas por el SW.



Pruebas de estrés

◊ Cómo se comporta el HW con cierta carga



Pruebas de seguridad

- ◊ Análisis estático de código
- ◊ Análisis dinámico de código
- ◊ Escáneres de seguridad externos
- ◊ Auditores de seguridad
- ◊ Pruebas manuales



QA

Diseño de pruebas

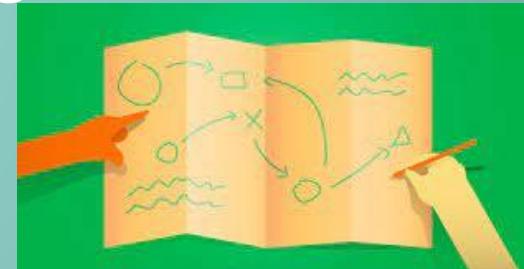


Las pruebas también necesitan
planificación y diseño



Plan de pruebas

- ◊ Definición del tipo de pruebas que se harán
- ◊ Definición de módulos que se probarán
- ◊ **Alcance de las pruebas**
- ◊ **Tiempo de ejecución de pruebas**



¿Por qué es importante el plan y la estrategia?

- ◊ Muchas veces es un entregable al cliente
- ◊ Es algo que ayuda a todo el equipo
- ◊ Los programadores también pueden ver las pruebas
- ◊ Las pruebas se ejecutarán de forma ordenada y se sabrá el estado de las mismas en todo momento
- ◊ Se pueden realizar iteraciones de forma clara



Es importante que las pruebas sean
flexibles y reutilizables

QA

Escenarios de prueba

h

- ◊ Creación de casos de pruebas
 - Definir el camino feliz
 - Definir casos negativos
- ◊ Definición de herramientas necesarias
- ◊ Definición de accesos necesarios
- ◊ **Comprendión detallada de los requerimientos**

Casos de prueba

- ◊ Título
- ◊ Pasos
- ◊ Resultados esperados
- ◊ Precondiciones
- ◊ Adjuntos

QA

Ejecución de pruebas

b

Pruebas Funcionales

- ◊ Verificar todas las combinaciones que un usuario podría ejecutar
- ◊ Ejecución de los escenarios de pruebas previamente diseñados

Estados de las pruebas

- ◊ Passed
- ◊ Failed
- ◊ Skip/Not applicable
- ◊ **Blocked**

X

Configure Chart

Chart Type



Name

Test Plan for Cycle 1 Overview

Group by*ⁱ

Outcome

Aggregationⁱ

Count

of

Tests

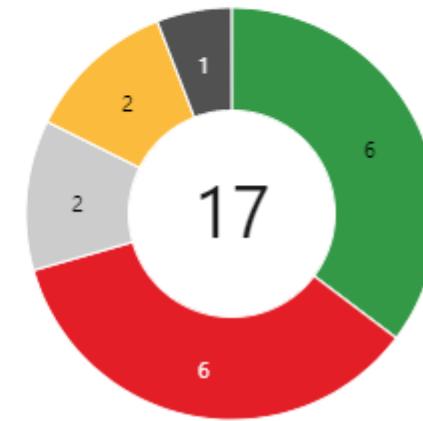
Sortⁱ

Value

Descending

Series

Passed Failed Paused In progress
Blocked



OK

Cancel

QA

Mejora continua

7

Pruebas como parte del proceso de mejora continua

- ◊ Ayuda a mejorar el software en cada iteración
- ◊ Entre más tipos de control de calidad se efectúen, el software se hace menos vulnerable.
- ◊ Las métricas provenientes de este proceso ayudan en otras fases
- ◊ Ayudan a los desarrolladores a mejorar

Mejora continua en las pruebas

- ◊ Al llevar un registro de todo se puede:
 - Mejorar en la creación de escenarios de prueba.
 - Agilizar el tiempo de ejecución de pruebas.
 - Agilizar la búsqueda de errores
 - Abarcar más tipos de prueba

Casos de prueba como métricas

- ◊ Nos indican la calidad de cada entrega del equipo.
- ◊ Nos indican un progreso del proyecto.
- ◊ Indican la factibilidad de una entrega.
- ◊ Se identifican errores recurrentes y puntos débiles en el proceso de desarrollo.

Administración de Recursos!

Del software

Qué puede ser un recurso?

Tiempo

El tiempo es un recurso sumamente crítico en el Desarrollo de cualquier Proyecto y se percibe como la cantidad de horas/días/semanas necesarias para realizar una tarea.

Recurso humano

El equipo presente en la ejecución de diferentes actividades para un Proyecto es un recurso también ya que permite separar las diferentes tareas de un Proyecto y dividirlas en base al nivel de especialización de los miembros del equipo.

Insumos

Diferentes proyectos necesitan distintos insumos para producir, en el software, se puede necesitar equipos de cómputo, infraestructura cloud, licencias, información

01.

Estimación y administración

Possible o imposible?



- No todo es posible en un Proyecto en un tiempo determinado.
- No todo es posible en Proyecto en base al recurso humano que se tiene.
- No todo es posible en un Proyecto en base al presupuesto de insumos que se tiene

Triángulo de hierro

Alcance

Qué se quiere lograr como parte del proyecto?

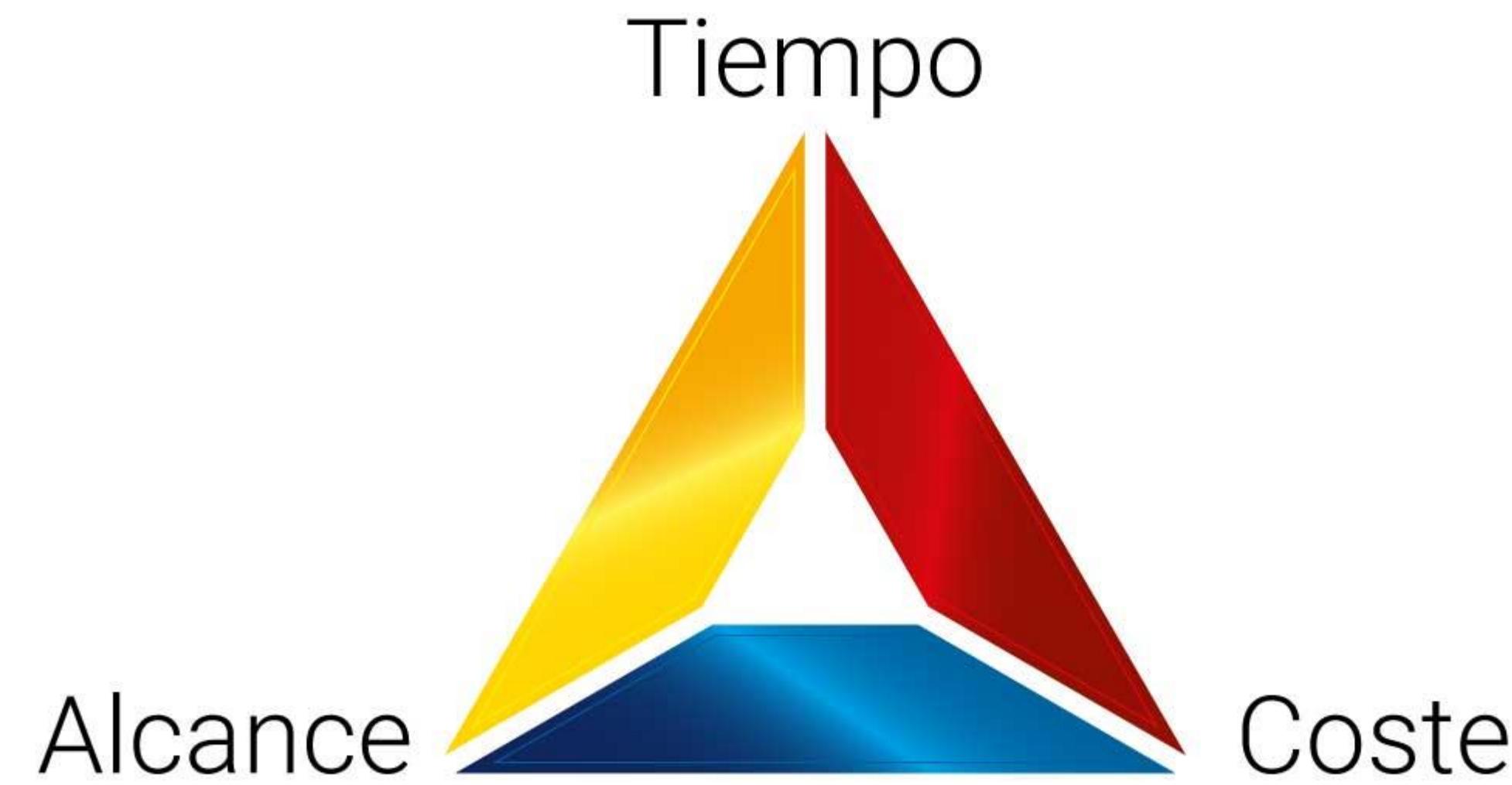
Tiempo

En cuánto tiempo se requiere terminar el Proyecto?

Recursos

Qué expertos, licencias o equipos se necesitan para llevar a cabo el proyecto?

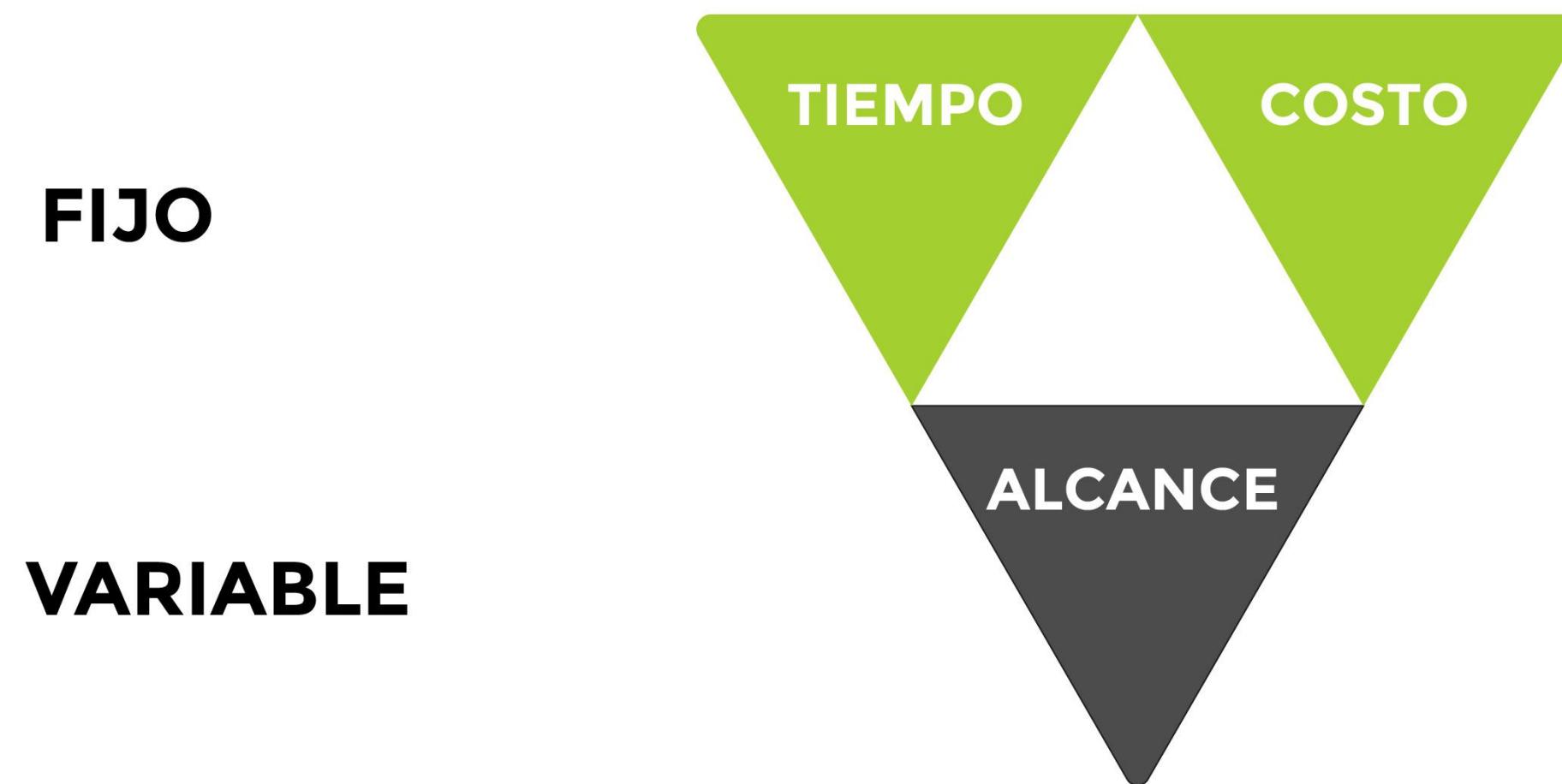
Triángulo de hierro



Triángulo de hierro



Triángulo de hierro



02.

Estimación de tiempos

Duración esperada



- Técnica de revisión y evaluación:

$$\frac{(\text{Tiempo optimista} + 4 \times \text{Tiempo más probable} + \text{tiempo Pesimista})}{6}$$

Otras técnicas de investigación

- Técnicas basadas en experiencia
- Técnicas basadas en modelo de algorítmico de costo

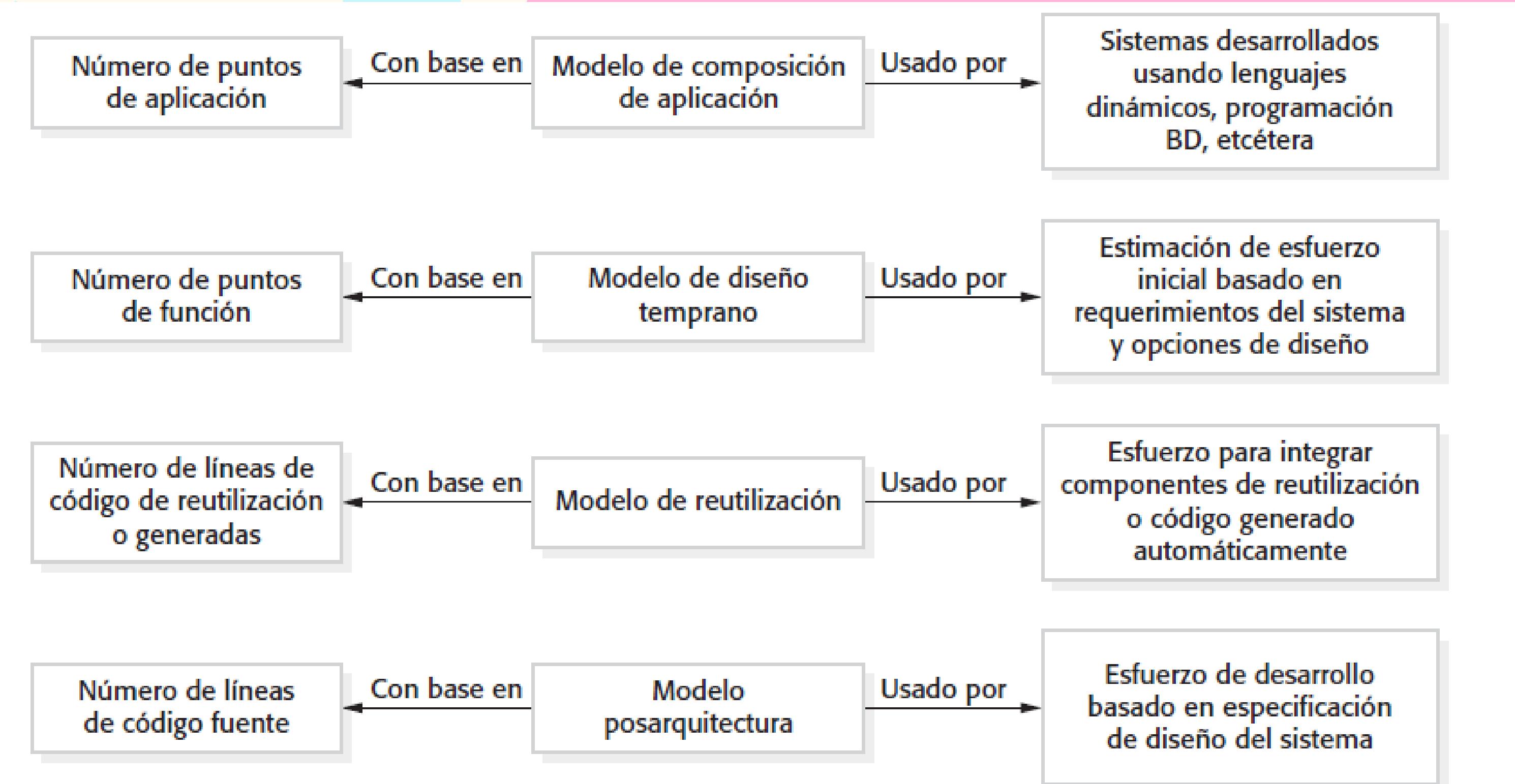
Técnicas basadas en experiencia

- La estimación se basa en experiencia con proyectos similares anteriores o con el dominio de la aplicación.

Técnicas basadas en el modelado algorítmico del costo

- Se basa en las estimaciones de los atributos del producto como su tamaño, características de proceso, o algún otro factor.
- $\text{Esfuerzo} = A * \text{Tamaño}^B * M$

En base a líneas de código



Modelo COCOMO II

- Modelo empírico
- Resultado de compilar datos de muchos proyectos de software
- Toma en cuenta el desarrollo en lenguajes dinámicos
- Toma en cuenta la integración de componentes

Submodelos de COCOMO II

1. Un modelo de composición
2. Diseño temprano
3. Modelo de Reutilización
4. Modelo postarquitectónica

Ejemplo COCOMO II

Experiencia y habilidad del desarrollador	Muy bajo	Bajo	Nominal	Alto	Muy alto
Madurez y capacidad ICASE	Muy bajo	Bajo	Nominal	Alto	Muy alto
PROD (NAP/mes)	4	7	13	25	50

Ejemplo COCOMO II

$$PM = (NAP \times (1 - \% \text{reutilización} / 100)) / PROD$$

$$PM_{Auto} = (ASLOC \times AT/100) / ATPROD // \text{Estimación para código generado}$$

Líneas de código

$$\text{ESLOC} = \text{ASLOC} \times \text{AAM}$$

ESLOC es el número equivalente de líneas de nuevo código fuente.

ASLOC es el número de líneas de código en los componentes que deben cambiarse.

AAM es un Multiplicador de Ajuste de Adaptación, como se estudia a continuación.

Calendarizar Tareas



Calendarizar Tareas

