Indice:

---

1. Herramientas del QA

1.1. Cobertura de tests

Cobertura: Porcentaje de código que esta cubierto por distintas pruebas (manuales o no). Es la cobertura de prueba del software para el que hacemos QA, nos dice cuánto código y funcionalidades se han probado.
La cobertura ayuda a definir entidades de sistema para poder cubrirlas con distintos tipos de pruebas.

- Que valor aporta?
    - Elimina defectos en etapas tempranas.
        - Ahorra problemas y costos.
        - Identifica discrepancias con especificaciones.
    - Cobertura para mas cobertura (mas satisfaccion cliente).
    - Elimina casos redundantes.
        - Tests sin sentido o inútiles.
        - Test Suites mas livianas y rápidas.
    - Mas ROI (Return on investment).
        - Menos defectos, mas ROI.
- Que beneficios tiene una alta cobertura?
    - Bastante filosofico.
    - Mas cobertura → mas codigo probado → Mas certeza de que funciona.
- Que porcentaje es sufuciente?
    - Del 80% al 90% segun proyecto y caracteristicas del area.
    - 100% dice solo que esta todo probado, no que se han probado situacion edge, por ejemplo.
- Que necesito para medir cobertura?
    - Escribir las pruebas.
    - Tener una herramienta para medir cobertura.
        - C# → MSTest
        - Visual Studio
        - Ruby → RSpec (escribir pruebas) y SimpleCov (medir cobertura)
        - Java
    - Herramienta de Testing.
        - Casos de uso.
        - Diagrama de clases.
        - Diagrama de entidad relacion.
        - Diagrama de secuencia.
        - Wireframes.
        - SQL - Base de datos.
        - Documentacion del sistema probado.
        - **Test Case.**

1.1.1. Tipos de cobertura

1.1.1.1. Statement coverage

Cubrir todos los condicionales o bifurcaciones del programa al menos una vez.
Nos proporciona detalles de:

- Bloques de código ejecutados y fallidos del total de bloques de código sin importar cuantos bucles, sentencias, lineas contenga.
- Crece exponencialmente el costo a medida que crece el N° de statements.

1.1.1.1.2. Decision/Branch coverage

- Se calcula encontrando el número mínimo de caminos que aseguren que todos los bordes han sido cubiertos.
- No hay una sola ruta que asegure la cobertura de la totalidad de aristas en una sola vez.

1.1.1.1.3. Path coverage

- Método de prueba estructural que implica encontrar todas las rutas ejecutables posibles.
- Para encontrar este “path” es necesario tener acceso al codigo.
- Garantiza la cobertura de todas las rutas de principio a fin y reduce la repeticion de paths anteriormente testeados.

1.1.1.1.4. Condition coverage

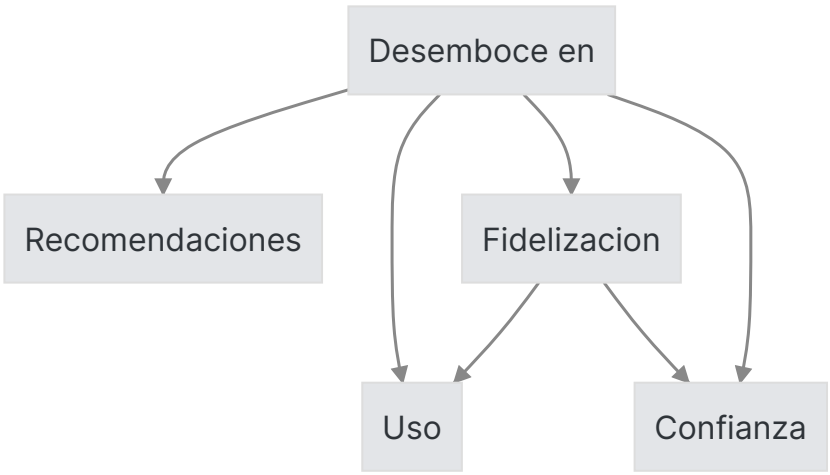
- Comprueba si se han ejercido ambos resultados (V o F) de cada condición.
- Basicamente intenta reducir la ejecucion mediante la evaluacion de las condiciones y asegurando de que cada una de las condiciones sea evaluada en sus valores “verdadero” y “falso” por lo menos una vez.
- Este tipo de testeo requiere dos casos de prueba por condición para dos resultados.

1.1.1.1.5. Boundary value coverage

- Muy útil para aquellas aplicaciones en las que se puede dar un error debido a los valores ingresados por el usuario o que fueron resultado de otro feature del mismo sistema.
- Cuando estos errores occurren en valores limites, es para lo que se usa el “boundary coverage”, los casos de prueba se seleccionan en los extremos de las clases de equivalencia.
- No es lo mismo **Boundary** que **Edge Case** ya que uno refiere a valores *limite* y el otro refiere a *escenarios* o *casos poco probables* de suceder que terminan sucediendo.

1.2. Navegabilidad

Facilidad con la que se pueden recorrer las distintas secciones de una pagina sin perderse.



1.2.1. Navegabilidad vs. Usabilidad vs. accesibilidad

Navegabilidad	**Usabilidad**	**Accesibilidad**
Facilidad para desplazarse entre los contenidos	Claridad, simpleza y estilo de la pagina	Diseño para ser usado por aquellos con capacidades diferentes

1.2.2. Preceptos y factores de la Navegabilidad

- Caracteristicas y componentes:
    - ***Menú de navegacion*** → Desplazamiento por las secciones prinsipales (por lo menos).
    - ***Ausencia de botones "Back"*** → Que no hayan "deadends", nunca la unica posibilidad puede ser volver. Es desagradable para el usuario.
    - ***Boton al "Home"*** → Permite volver al inicio (reiniciar la navegacion) en caso de haberse perdido, por ejemplo. Debe estar presente en todas las paginas del sitio.
    - ***Regla 3 clicks*** → El usuario de una web debe ser capaz de encontrar cualquier informacion con no mas de eso. No siempre se puede. Lo "ideal" es 1 click je.
- Factores:
    - ***Robustez:***
        - Grado en que el programa maneja entradas erroneas o interacciones inapropiadas.
        - Reconoce los errores en el limite? Y mas alla?
        - Continua operando sin fallar o degradarse?
        - Debe guiar en forma explisita al usuario para que vuelva al camino correcto.
        - La interfaz debe dar un diagnostico y guía útiles.

- ***Riqueza:***
    - Grado en que el programa provee un conjunto abundante de características.
    - Personalizacion de la interfaz segun la necesidad del usuario.

1.3. Dilemas de la Calidad de Software

1.3.1. Punto medio mágico (Bertrand Meyer)

- Mal software → Nadie lo quiere usar → Pérdida
- Software perfecto → Mucho tiempo y muy caro (Gran esfuerzo) → Queda fuera del negocio → Pérdida

Punto medio mágico ⟹ Suficientemente bueno

1.3.2. Software "suficientemente bueno"

- Tiene funciones y caracteristicas de alta calidad deseables por usuarios.
- Tiene funciones y caracteristicas ***oscuras*** y especializadas con errores conocidos.

Esperamos que nos perdonen los errores gracias a una alta conformidad con las demas funcionalidades de la aplicacion.

1.3.3. El costo de la calidad

Calidad ⟹ Tiempo y dinero (Mucho, demasiado...)
Mala calidad ⟹ Tiempo y dinero (Mucho, demasiado...) → Los usuarios lo sufren y los desarrolladores lo mantienen.

1.4.Niveles de staging

- Desde el punto de vista del QA es en un entorno de prueba (stage).

    - **Réplicas casi exactas** de un entorno de produccion pero para pruebas (para no dañar o arriesgar los datos de los usuarios).
        - Configuraciones de hardware.
        - Servidores.
        - Bases de datos.
        - Cache.
    - Diseñados para probar y garantizar la calidad un entorno similar al de producción antes de implementar (hacer una release).
        - Código.
        - Funcionalidades
        - Compilaciones.

        - Versiones.
        - Actualizaciones.

- Cantidad de niveles varia segun:

    - Tamaño.
    - Infraestructura.
    - Dinero.
    - Criticidad de realizar las pruebas.

- Se necesita un nivel de Staging aparte solo para QA para poder tener mas autonomia en su manejo y entonces mejor resultado va a tener el trabajo del equipo.

- Niveles minimos:

    - Stage 1: Local dev.
    - Stage 2: Common Dev.
    - Stage 3: QA / UAT.
    - Stage 4: Production (Este no salia pero medio que si tiene que estar, aunque no es "Staging").

1.4.1. Tipos de pruebas de "Staging"

- Smoke testing:
    - Verifica funcionalidades esenciales del servicio, que no se haya roto nada que antes andaba.
- Pruebas de aceptacion del usuario (UAT):
    - Realizada desde la perspectiva del usuario para garantizar calidad.
- Ingenieria del caos:
    - Refuerza la confianza en el desarrollo al intentar romper el código constantemente.

---

2. Reportes de errores: Metricas y Objetivos

2.1. Errores y bugs

Bug: Fallo, error o problema que provoca o desencadena en un resultado indeseado.

- No todo defecto es por código malo.
- Fuente comun de defectos: **falta de requerimientos** o requerimientos pobres.

- 1 error puede desencadenar varios sintomas.

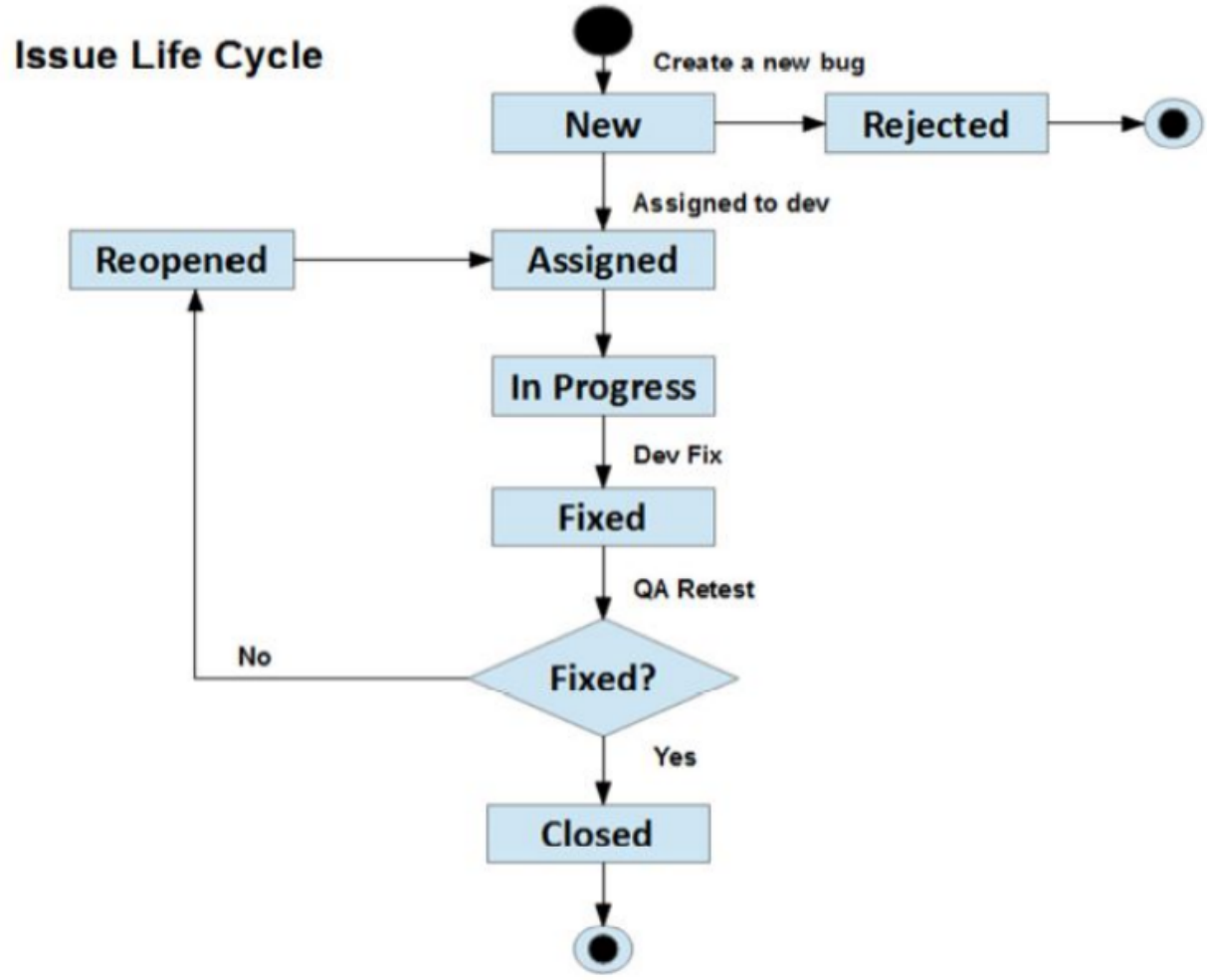
Error programador → Códgo malo → Ejecucion de ocacion con error

2.1.1 Reporte de bugs

Con **bug trackers** como Jira, redmine, mantis. Utilizan sistemas de tickets.

- Pautas a cumplir del reporte:
  - Asociado a un proyecto.
  - ID.
  - Descipcion completa explicando el bug.
  - Pasos para reproducirlo.
  - Resultado esperado.
  - Resultado real.
  - Entorno de prueba (OS, browsers, etc).
  - Screenshots, videos.

2.1.2. Ciclo de vida de un bug



2.2. Metodos de prueba

2.2.1. Pruebas estáticas

- Basadas en:
    - Examen manual o revision.
    - Analisis automatizado o analisis estático de codigo.
    - Documentacion del codigo.

Revisiones: Forma de probar los productos de trabajo del software y codigo que pueden realizarse antes de ejecutar las pruebas dinamicas. Ej: peer reviews.

Beneficios
Encontramos defectos rapido y pronto, mientras son baratos de resolver.
Desarrollo de mejoras en el proceso y la productividad.
Ahorro de tiempo y dinero.

Defectos faciles de encontrar con este metodo
Desviaciones de los estandares.
Defectos de requerimientos.
Defectos de diseño, mantenibilidad o interfaz.

- Productos susceptibles a revision:
    - Requerimientos.
    - Código.
    - Test cases.
    - Scripts.
    - Bugs reportados.

2.2.2. Pruebas dinámicas

Conjunto de pruebas que requiere la ejecucion (Compilacion y ejecucion) del código para determinar su funcionamiento a lo largo del tiempo

- Pruebas que se pueden hacer:
    - Pruebas unitarias
    - Pruebas de integracion
    - Pruebas de sistema
    - Pruebas de aceptacion

2.3.ISO 9001

- Quiere establecer los elementos basicos de un sistema de gestion de la calidad:
    - PDCA.

        - Define politica que muestra donde esta el enfasis del sistema.
- Documentar el sistema de calidad:
    - Describir el proceso.
    - Producir un manual de operacion.
    - Desarrollar metodos para controlar / actualizar documentos.
    - Establecer metodos de registro.
- Apoyar el control y aseguramiento de calidad:
    - Promover la importacia de la calidad en todos.
    - Centrarse en satisfacer al cliente.
    - Crear un plan de calidad acorde a los objetivos, responsabilidades y autoridad.
    - Definir mecanismos de comunicacion entre participantes.
- Establecer mecanismos de revisión para el sistema de administración de la calidad:
    - Métodos de revisión y mecanismos de retroalimentación.
    - Definir procedimientos para dar seguimiento.
- Identificar recursos para la calidad (personal, capacitación, infraestructura, etc):
    - Establecer mecanismos de:
        - Control.
        - Planeación.
        - Requerimientos del cliente.
        - Actividades técnicas (análisis, diseño y pruebas).
        - Vigilancia y administración del proyecto.
- Definir métodos de corrección:
    - Evaluar datos y métricas de la calidad
    - Definir el enfoque para la mejora continua del proceso y la calidad.

2.4. Objective and Key Results (OKRs)

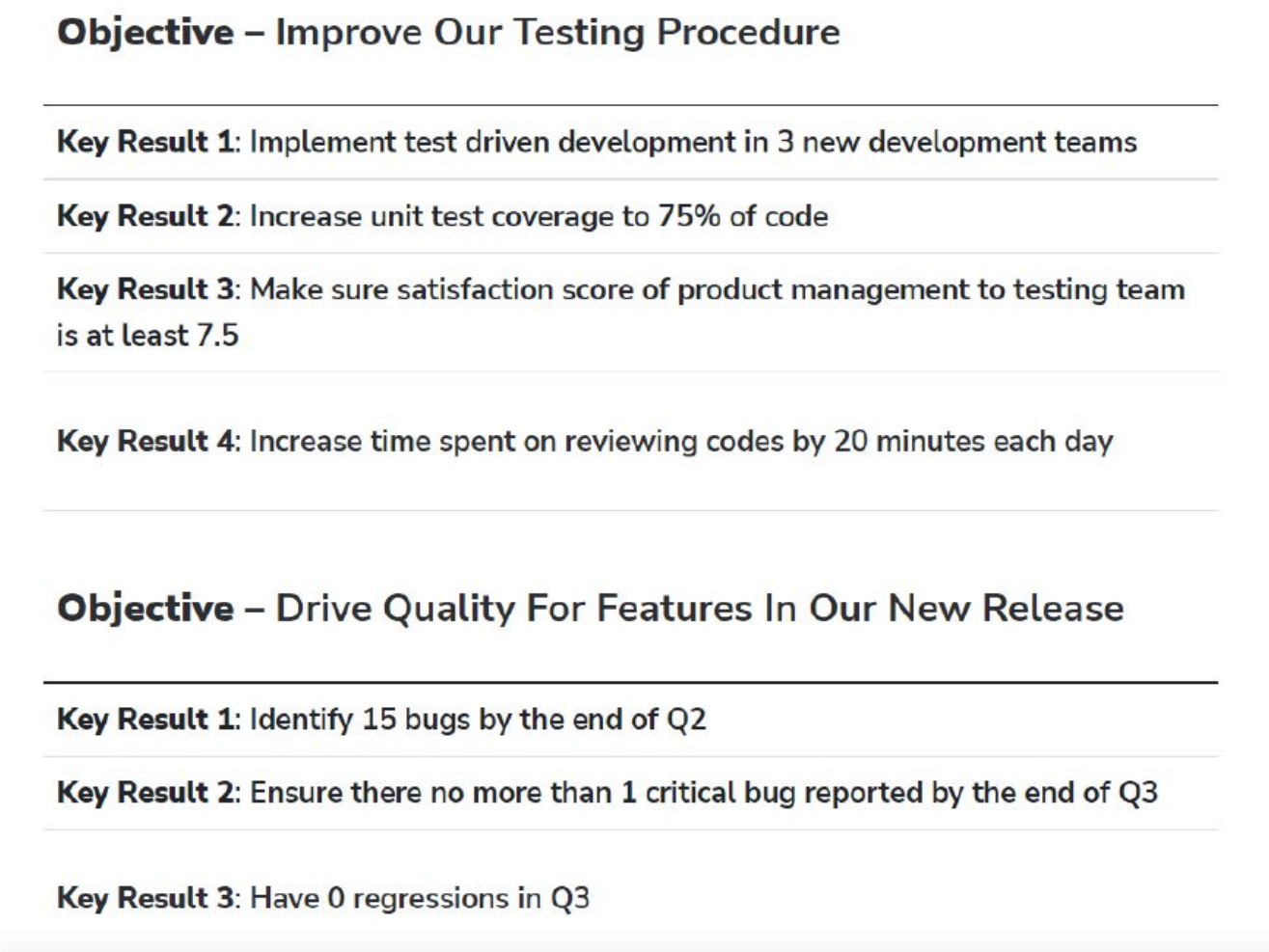
Framework / metodologia de objetivos o desafios impuestos por equipos y personas para lograr crecimiento y mejora en diferentes ambitos:

- Personal.
- Laboral.
- Economico.
- Profesional.

Se diferencian de herramientas de productividad (como TODOs) porque no son acciones, sinó los resultados de las mismas, no son aspectos "obvios a simple vista".
Se define un Objetivo (O) y despues sus resultados clave (KRs).
Objective: Metas aspiracionales no especificas de un problema a resolver.
Key Results: Medibles en algun valor y en tiempo. Son los que se miden para saber si se está cumpliendo el *objetivo*.
Es una alternativa simple, flexible y eficaz para que la organización mantenga foco en

los temas prioritarios.
Se pueden interpretar como la fragmentacion de los objetivos estrategicos de una organizacion en tareas mas concretas, cortas, desafiantes y medibles. Es un cambio de paradigma, ya que rigen las iniciativas a lo largo del periodo por el que fueron definidas.

Ejemplo para un team de calidad:



Beneficios	Descripción

Beneficios	Descripción
Comunicacion:	Sistemas faciles de entender favorecen el acuerdo y uso del sistema.
Agilidad:	Ciclos de frecuencia ágil, preparado para el cambio.
Foco:	Deja a todos claro lo que mas importa.
Colaboracion:	Promueve alineacion funcional cruzada. Co-responsabilidad de objetivos.
Transparencia:	Visibiliza metas transversalmente. Da claridad.
Compromiso:	Son creados de forma participativa para que las personas y equipos sean dueños de sus objetivos.
Visionario:	Extiende la idea de lo que es posible.

2.4.1. Key Performance Indicators (KPIs)

Es una meta en forma de un valor medible, una métrica, un numero. Nos ayuda a comparar un parametro (el KPI) con un valor preestablecido.

2.4.1.1. OKRs vs. KPIs

OKR	KPI
Objetivo	Metrica
Ej: "O: Aumentar la retencion de clientes"	Ej: "Tasa de abandono"
Ej: "KR: Tasa de abandono en un 15%"	