## Testing:

* Proceso destructivo de tratar de encontrar defectos (cuya presencia se asume) en el código.
* Se debe ir con una actitud negativa para demostrar que algo es incorrecto.
* Busca demostrar que el software hace lo que tiene que hacer y de la manera correcta.
* Es destructivo porque busca romper el software, busca romper lo que se está probando para encontrar los lugares donde falla.
* Testing exitoso: es el que encuentra defectos.
* Mundialmente: 30 a 50% del costo de un software confiable.

Gráfico, Diagrama

Descripción generada automáticamente

Está dentro de las actividades para el aseguramiento de calidad de software.

Auditoria de proyecto = si el proyecto está cumpliendo lo que dijo.

Auditoria de calidad = quien contrala a quien hace calidad para ver si hace calidad.

Revisiones técnicas = son entre pares. No son jerárquicas. No puede ser la gente de calidad ni el jefe. Son para mejorar algún artefacto.

El testing incluye las 2 V: verificación y validación.

* Verificación: ¿Estamos construyendo el sistema correctamente?
* Validación: ¿Estamos construyendo el sistema correcto?. Se hace foco en las pruebas de aceptación porque estamos validando con el cliente que el software cumpla con los requerimientos.

### Asegurar la calidad vs controlar la calidad:

Una vez definidos los requerimientos de calidad, debo tener en cuenta que:

* La calidad no puede “inyectarse” al final.
* La calidad del producto depende de tareas realizadas durante todo el proceso.
* Detectar errores en forma temprana ahorra esfuerzos, tiempo y recursos.
* La calidad no solamente abarca aspectos del producto sino también del proceso y como éstos se pueden mejorar, que a su vez evita defectos recurrentes.
* El testing NO puede asegurar ni calidad en el software ni software de calidad.

El aseguramiento de calidad implica detección temprana, tratar de prevenir. La prevención es más barata.

### Conceptos error vs defecto:

Error cuando se lo identifica antes de la etapa siguiente.

Como el testing es la última etapa ya estamos hablando de defectos.

Los errores de programación son corregibles, los errores de requerimientos no son corregibles. Algunos errores de arquitectura son corregibles.

No todos los errores y defectos provocan fallas. Las fallas son un mal funcionamiento de sistema.

El defecto es el que se encuentra en una etapa posterior a la que se inyecto, se encuentra en producción. Es más caro corregir un defecto que un error.

El error es el que se encuentra en una etapa anterior a enviarlo a producción.

Si en una prueba unitaria encuentro fallas es un error.

Las metodologías ágiles buscan, entre otras cosas, reducir el tiempo entre la definición de los requerimientos y las fases de desarrollo y pruebas. Al acortar este intervalo, cualquier falla detectada puede corregirse de manera más rápida y económica.

### Conceptos defectos, severidad y prioridad:

El testing exhaustivo (testear el 100%) es imposible. Para esto hay algunos criterios para priorizar.

Severidad: determina que tan grave es el defecto. La asigna la gente de testing.

1. Bloqueante. Es cuando el sistema no anda. Impide el funcionamiento del sistema.
2. Crítico. Provoca una falla, pero no una falla general, es una falla en alguna funcionalidad.
3. Mayor. Provoca una falla, pero nos permite seguir trabajando.
4. Menor. No impide la continuidad del sistema.
5. Cosmético. Es un error o defecto que no provoca falla (mal funcionamiento) y que normalmente tiene que ver con aspectos estéticos.

Prioridad: es del lado del negocio.

1. Urgente.
2. Alta.
3. Media.
4. Baja.

### Niveles de prueba:

Forma

Descripción generada automáticamente con confianza media

#### Testing unitario:

* Se prueba cada componente tras su realización/construcción.
* Solo se prueban componentes individuales.
* Cada componente es probado de forma independiente.
* Se produce con acceso al código bajo pruebas y con el apoyo del entorno de desarrollo, tales como un framework de pruebas unitarias o herramientas de depuración.
* Los errores se suelen reparar tan pronto como se encuentran, sin constancia oficial de los incidentes.

#### Testing de integración:

* Test orientado a verificar que las partes de un sistema que funcionan bien aisladamente, también lo hacen en conjunto.
* Cualquier estrategia de prueba de versión o de integración debe ser incremental, para lo que existen dos esquemas principales:
  + Integración de arriba hacia abajo (top-down).
  + Integración de abajo hacia arriba (bottom-up).
* Lo ideal es una combinación de ambos esquemas.
* Tener en cuenta que los módulos críticos deben ser probados lo más tempranamente posible.
* Los puntos clave del test de integración son simples:
  + Conectar de a poco las partes más complejas.
  + Minimizar la necesidad de programas auxiliares.

#### Testing de sistema:

* Es la prueba realizada cuando una aplicación está funcionando como un todo (prueba de la construcción final).
* Trata de determinar si el sistema en su globalidad opera satisfactoriamente (recuperación de fallas, seguridad y protección, stress, performance, etc.).
* El entorno de prueba debe corresponder al entorno de producción tanto como sea posible para reducir al mínimo el riesgo de incidentes debidos al ambiente específicamente y que no se encontraron en las pruebas.
* Deben investigar tanto requerimientos funcionales y no funcionales del sistema.

#### Testing de aceptación de usuario:

* Es la prueba realizada por el usuario para determinar si la aplicación se ajusta a sus necesidades.
* La meta en las pruebas de aceptación es el de establecer confianza en el sistema, las partes del sistema o las características específicas y no funcionales del sistema.
* Encontrar defectos no es el foco principal en las pruebas de aceptación.
* Comprende tanto la prueba realizada por el usuario en ambiente de laboratorio (pruebas alfa), como la prueba en ambientes de trabajo reales (pruebas beta).

### Ambientes para construcción del software:

* Desarrollo. Las pruebas unitarias se hacen acá.
* Prueba. Las pruebas de sistemas se hacen acá.
* Pre-producción. Las pruebas de aceptación se hacen acá, pero sino existe se hacen en producción.
* Producción.

Un ambiente es un espacio de trabajo. No se debería hacer pruebas en el ambiente de producción.

Se deben cumplir al menos: desarrollo, prueba y producción.

### Caso de prueba:

Es lo que diseña el analista de pruebas. Describe indicaciones paso a paso de como ejecutar el sistema en una funcionalidad en particular.

Un caso de prueba es la unidad de la actividad de la prueba.

Es una de las salidas de proceso de testing.

se define para ejecutar el mismo conjunto de pasos bajo las mismas condiciones tantas veces como sea necesario, con el fin de verificar si los defectos detectados han sido corregidos.

Consta de 3 partes:

1. Objetivo: la característica del sistema a comprobar.
2. Datos de entrada y de ambiente: datos a introducir al sistema que se encuentra en condiciones preestablecidas.
3. Comportamiento esperado: la salida o la acción esperada en el sistema de acuerdo a los requerimientos del sistema.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Escala de tiempo

Descripción generada automáticamente

* Set de condiciones o variables bajo las cuales un tester determinará si el software está funcionando correctamente o no.
* Buena definición de casos de prueba nos ayuda a reproducir defectos.
* “Los bugs se esconden en las esquinas y se congregan en los límites…”
* Objetivo: descubrir errores.
* Criterio: en forma completa.
* Restricción: con el mínimo de esfuerzo y tiempo.

### Derivación de casos de prueba:

Escala de tiempo

Descripción generada automáticamente con confianza media

### Conclusiones sobre la generación de casos:

* Ninguna técnica es completa.
* Las técnicas atacan distintos problemas.
* Lo mejor es combinar varias de estas técnicas para complementar las ventajas de cada una.
* Depende del código a testear.
* Sin requerimientos todo es mucho más difícil.
* Tener en cuenta la conjetura de defectos.
* Ser sistemático y documentar las suposiciones sobre el comportamiento o el modelo de fallas.

### Condiciones de prueba:

* Esta es la reacción esperada de un sistema frente a un estímulo particular, este estímulo está constituido por las distintas entradas.
* Una condición de prueba debe ser probada por al menos un caso de prueba.

### Estrategias de prueba:

Caja blanca y caja negra.

Se usan para determinar cuántos casos de prueba se van a necesitar para cada funcionalidad.

Buscan como hago para cubrir la mayoría de las cosas que tengo que probar.

Sirven para lograr la mayor cobertura de prueba (maximizar la cantidad de defectos encontrados) con el menor corto posible (en termino de plata, esfuerzo, etc.).

En el método de caja blanca se tiene acceso al código, mientras que en el método de caja negra se conocen los parámetros de entrada y las salidas esperadas.

#### Métodos:

* ¿Para qué usarlos? El tiempo y el presupuesto es limitado.
* Hay que pasar por la mayor cantidad de funcionalidades con la menor cantidad de pruebas.

#### Caja negra:

Las entradas y salidas están identificadas pero el proceso se desconoce.

Es con el sistema funcionando.

Son pruebas dinámicas, porque se ejecuta el software.

Hay 2 métodos de testing de caja negra:

* Basado en especificaciones.
  + Partición de equivalencias.
  + Análisis de valores límites.
  + Etc.
* Basados en la experiencia.
  + Adivinanza de defectos: ir a buscar el defecto porque se sabe que está ahí (por la experiencia se sabe).
  + Testing exploratorio: es sentarse a probar.

Basado en la experiencia: es basado en la experiencia de quien va a ejecutar el testing. No lo puede hacer cualquiera, porque lo tiene que hacer alguien que tenga experiencia.

##### Partición de equivalencias:

Imagen que contiene Logotipo

Descripción generada automáticamente

Una clase de equivalencia es un subconjunto de valores que cumplen con una condición (que se llama condición externa).

Se define una condición externa y de ahí, se definen las clases de equivalencias vinculadas con esa condición externa.

El concepto de equivalencia en el método de partición de equivalencia se refiere a que se asume que todos los valores dentro de una misma clase de equivalencia producirán un el resultado equivalente. Si funciona para uno de los valores, asumo que funciona para ese conjunto de valores.

Dos pasos:

1. Identificar las clases de equivalencia (válidas y no válidas).
   1. Rango de valores continuos.
   2. Valores discretos.
   3. Selección simple.
   4. Selección múltiple.
2. Identificar los casos de prueba.

##### Análisis de valores límites:

* Es una variante de la partición de equivalencias, en vez de seleccionar cualquier elemento como representativo de una clase de equivalencia, se seleccionan los bordes de una clase.

Utiliza solamente los valores extremos para probar.

No siempre va a existir valores límites, porque no siempre se trabaja con límites de fechas o numéricos.

##### Basados en experiencia:

* Adivinanza de defectos.
* Testing exploratorio.

#### Caja blanca:

* Se basan en el análisis de la estructura interna del software o un componente del software.
* Se puede garantizar el testing coverage.

Texto

Descripción generada automáticamente

Son pruebas estáticas, porque no se ejecuta el software para probar.

Se hacen en el código.

Se quiere ver la calidad del código, código malicioso, etc.

Clase de equivalencia = conjunto de casos de prueba -> condiciones externas.

Clase de equivalencias de entradas (cuales son los atributos que definen las entradas) y de salida (las salidas esperadas).

Solo hay una condición válida para una clase de equivalencia por ejemplo edad >= 18. Pero puede tener varias condiciones inválidas.

El debug y el testing de caja blanca no son lo mismo porque el debug es dinámico porque se debe ejecutar el software, además, el objetivo del debug es encontrar donde está el defecto (porque ya se sabe de la presencia del mismo); con el testing se busca si hay defectos, no se sabe si hay defectos.

### Ciclo de test:

Un ciclo de pruebas abarca la ejecución de la totalidad de los casos de prueba establecidos aplicados a una misma versión del sistema a probar.

El primer ciclo de prueba se llama ciclo 0.

Se debe tener idealmente 2 ciclos de pruebas (sino hay errores o defectos).

### Regresión:

Al concluir un ciclo de pruebas, y reemplazarse la versión del sistema sometido al mismo, debe realizarse una verificación total de la nueva versión, a fin de prevenir la introducción de nuevos defectos al intentar solucionar los defectos.

Es una estrategia de cómo se va a encarar el testing a partir del ciclo 1.

Sin regresión significa que solo se prueba de la lista de defectos.

Captura de pantalla de un celular

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Captura de pantalla de un celular con letras

Descripción generada automáticamente

### Proceso de pruebas:

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

#### Planificación y control:

* La planificación de las pruebas es la actividad de verificar que se entienden las metas y los objetivos del cliente, las partes interesadas (stakeholders), el proyecto, y los riesgos de la pruebas que se pretende abordar.
* Construcción del test plan:
  + Riesgos y objetivos del testing.
  + Estretegia de testing.
  + Recursos.
  + Criterio de aceptación.
* Controlar:
  + Revisar los resultados del testing.
  + Test converage y criterio de aceptación.
  + Tomar decisiones.

#### Identificación y especificación:

* Revisión de la base de pruebas.
* Verificación de las especificaciones para el software bajo pruebas.
* Evaluar la testeabilidad de los requerimientos y el sistema.
* Identificar los datos necesarios.
* Diseño y priorización de los casos de pruebas.
* Diseño del entorno de prueba.

#### Ejecución:

* Desarrollar y dar prioridad a nuestros casos de prueba.
* Crear los datos de prueba.
* Automatizar lo que sea necesario.
* Creación de conjuntos de pruebas de los casos de prueba para la ejecución de la prueba eficientemente.
* Implementar y verificar el ambiente.
* Ejecutar los casos de prueba.
* Registrar el resultado de la ejecución de pruebas y registrar la identidad y las versiones del software en las herramientas de pruebas.
* Comparar los resultados reales con los resultados esperados.

#### Evaluación y reporte:

* Evaluar los criterios de aceptación.
* Reporte de los resultados de las pruebas para los stakeholders.
* Recolección de la información de las actividades de prueba completadas para consolidad.
* Verificación de los entregables y que los defectos hayan sido corregidos.
* Evaluación de cómo resultaron las actividades de testing y se analizan las lecciones aprendidas.

Icono

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

### El testing en el ciclo de vida del software:

Objetivos de involucrar las actividades de testing de manera temprana:

* Dar visibilidad de manera temprana al equipo, de cómo se va a probar el producto.
* Disminuir los costos de correcciones de defectos.

#### Modelo en V:

Diagrama

Descripción generada automáticamente

### Romper mitos:

* El testing es una etapa que comienza al terminar de codificar.
* El testing es probar que el software funciona.
* Testing = calidad de producto.
* Testing = calidad de proceso.
* El tester es el enemigo del programador.

### ¿Cuánto testing es suficiente?

* El testing exhaustivo es imposible.
* Decidir cuánto testing es suficiente depende de:
  + Evaluación del nivel de riesgo.
  + Costos asociados al proyecto.
* Usamos los riesgos para determinar:
  + Qué testear primero.
  + A qué dedicarle más esfuerzo de testing.
  + Qué no testear (por ahora).
* El criterio de aceptación es lo que comúnmente se usa para resolver el problema de determinar cuándo una determinada fase de testing ha sido completada.
* Puede ser definido en términos de:
  + Costos.
  + % de tests corridos sin fallas.
  + Fallas predichas aún permanencen en el software.
  + No hay defectos de una determinada severidad en el software.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

### Principios del testing:

* El testing muestra presencia de defecto.
* El testing exhaustivo es imposible.
* Testing temprano.
* Agrupamiento de defectos.
* Paradoja del presticida.
* El testing es dependiente del contexto.
* Falacida de la ausencia de errores.
* Un programador debería evitar probar su propio código.
* Una unidad de programación no debería probar sus propios desarrollos.
* Examinar el software para probar que no hace lo que se supone que debería hacer es la mitad de la batalla, la otra mitad es ver que hace lo que no se supone que debería hacer.
* No planificar el esfuerzo de testing sobre la suposición de que no se van a encontrar defectos.

### La psicología del testing:

* La búsqueda de fallas puede ser visto como una crítica al producto y/o su autor.
* La construcción del software requiere otra mentalidad a la de testear el software.

### Smoke test:

* Primera corrida de los test de sistema que provee cierto aseguramiento de que el software que está siendo probado no provoca una falla catastrófica.

### Tipos de pruebas:

* Testing funcional:
  + Las pruebas se basan en funciones y características (descripta en los documentos o entendidas por los testers) y su interoperabilidad con sistemas específicos.
    - Basado en requerimientos.
    - Basado en los procesos de negocio.
* Testing no funcional:
  + Es la prueba de “cómo” funciona el sistema.
  + No hay que olvidarlas. Los requerimientos no funcionales son tan importantes como los funcionales.
    - Performance testing.
    - Pruebas de carga.
    - Pruebas de stress.
    - Pruebas de usabilidad.
    - Pruebas de mantenimiento.
    - Pruebas de fiabilidad.
    - Pruebas de portabilidad.

### Pruebas de interfaces de usuarios:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

### Prueba de performance:

* Tiempo de respuesta.
* Concurrencia.

### Prueba de configuración:

Imagen que contiene Texto

Descripción generada automáticamente

### TDD:

“El acto de diseñar tests es uno de los mecanismos conocidos más efectivos para prevenir errores… El proceso mental que debe desarrollarse para crear test útiles puede descubrir y eliminar problemas en todas las etapas del desarrollo”.

* Desarrollo guiado por pruebas de software, o Test-Driver Development (TDD).
* Es una técnica avanzada que involucra otras dos prácticas: escribir las pruebas primero (Test First Development) y refactorización (Refactoring).
* Para escribir las pruebas generalmente se utilizan las pruebas unitarias.