



Aseguramiento de la Calidad

Julieta Barrionuevo
Universidad de Mendoza

Contenidos

- ❑ Introducción a la calidad del software.
- ❑ SDLC (software Development Life Cycle)
- ❑ Aseguramiento de la Calidad
- ❑ Testing
- ❑ Proceso de Testing
- ❑ Pruebas de Testing
- ❑ Rol del tester
- ❑ Ambientes de Prueba
- ❑ Herramientas de Prueba
- ❑ Defectos-Bugs
- ❑ Pruebas de Caja Negra y Caja Blanca
- ❑ Pruebas de unidad
- ❑ Tipos de Testing

Contenidos (Edited)

- ❑ Introducción a la calidad del software.
- ❑ SDLC (Software development life cylce)
- ❑ Que es Aseguramiento de la Calidad de software
- ❑ Tipos de Testing
 - ❑ Pruebas de Caja Negra y Caja Blanca
 - ❑ Pruebas de unidad
 - ❑ Integracion, Performance, Carga
- ❑ Procesos y Pruebas de Calidad de software
- ❑ Rol y responsabilidades del tester
- ❑ Ambientes de Prueba (Staging)
- ❑ Herramientas y tipos de Prueba
- ❑ Deteccion y reporte de Defectos-Bugs

Objetivos

- Incorporar **Conceptos generales** sobre calidad de sw. y su significado
- Conocer las **competencias** del profesional de la calidad de Software
- Comprender la **Importancia** de trabajar para y con Calidad (Mitos y Realidades)
- Comprender de qué manera se llevan a cabo **revisiones de software** eficaces.
- Comprender el valor del area de QA en los distintos tipos de metodologías ágiles
- Adquirir destrezas para la creación de casos de **prueba eficaces**.
- Adquirir destrezas para **testeo de software** utilizando distintas metodologías.
- Integrar conocimientos de **análisis de sistemas y metodologías Ágiles**
- Obtener conocimiento para proponer/comprender estrategias de QA
- Conocer **herramientas de Prueba** y el uso efectivo de las mismas

Inserción en el Mercado Laboral

- Conocer conceptos sobre **Mejora de procesos**
- Prepararnos para la Demanda del Mercado.

Requisitos para obtener la Promoción/Regularidad:

Polinomio = Promedio (Trabajos Prácticos) * **0.2** +
Promedio(Exámenes Parciales) * **0.3** + Asistencia * **0.1** +
Proyecto Final * **0.4**

- **Promoción** = Polinomio ≥ 6
- **Regularidad** = $4 \leq$ Polinomio < 6
- **Recursado** = Polinomio < 4

Importa la Calidad de software?

Un ejemplo extremo:

En el mes de noviembre de 2000, en un hospital de Panamá, 28 pacientes recibieron dosis masivas de rayos gama durante su tratamiento contra diversos tipos de cáncer. En los meses que siguieron, 5 de estos pacientes murieron por envenenamiento radiactivo y 15 más sufrieron complicaciones serias. ¿Qué fue lo que ocasionó esta tragedia? Un paquete de software, desarrollado por una compañía estadounidense, que fue modificado por técnicos del hospital para calcular las dosis de radiación para cada paciente.

Los tres médicos panameños que "customizaron" el software para que diera capacidad adicional fueron acusados de asesinato en segundo grado. La empresa de Estados Unidos enfrentó litigios serios en los dos países. Gage y McCormick comentan lo siguiente: Éste no es un relato para prevenir a los médicos, aun cuando luchan por estar fuera de la cárcel si no entienden o hacen mal uso de la tecnología. Tampoco es la narración de cómo pueden salir heridos, o algo peor, los seres humanos a causa del software mal diseñado o poco explicado, aunque hay muchos ejemplos al respecto.

CAPITULO 1



Introducción a la calidad del software.

Software con calidad?

- ¿Hace lo que se le pide?
- ¿Lo hace de forma **confiable** todo el tiempo?
- ¿Utiliza los recursos adecuados para la ejecución de sus funciones?
- ¿Pueden prevenirse accesos no autorizados a la información que maneja?
- ¿Es **fácil** y **cómodo** de manejar?



Un poco de Historia...

▣ 1990:

- ▣ Las principales corporaciones desperdiciaban miles de millones de dólares en software que no tenía las características ni la funcionalidad que se habían prometido.

▣ CIO Magazine [Lev01]

Dió la alerta: “**Dejemos de desperdiciar \$78 mil millones de dólares al año**”, y lamentaba el hecho de que “**las empresas estadounidenses gastan miles de millones de dólares en software que no hace lo que se supone que debe hacer**”.

¿Cuán malo es el software defectuoso?

- Los expertos dicen que sólo se requiere de **3 a 4** cuatro defectos por cada **1000** líneas de código para que un programa tenga mal desempeño.
- La mayoría de los programadores cometen **1** error en cada **10** líneas de código que escriben, lo que, multiplicado por los millones de líneas....

Conclusión:

La corrección de los errores cuesta a los vendedores de software al menos la mitad de sus presupuestos de desarrollo durante las pruebas.

CALIDAD ¿Moda o Necesidad?

- Todas las empresas quieren producir aplicaciones informáticas con alta calidad, en el **menor tiempo posible** y a **costos mínimos**.
- Los clientes son más **exigentes** requieren que se satisfaga sus **expectativas**
- Las empresas se están enfrentando a un público que, lejos de ser conformista, espera de su proveedor resultados con altos niveles de **calidad**
- La **calidad del software** juega un papel importante dentro del desarrollo de aplicaciones informáticas influyendo positivamente en la decisión de un cliente a la hora de escoger el producto que necesita
- Empresas necesitan ser **Competitivas**





Qué entendemos por
calidad de Software?

¿QUÉ ES CALIDAD?

David Garvin , de Harvard Business School,



□ El **punto de vista trascendental** :

- El *punto de vista del usuario* que concibe la calidad en términos de las metas específicas del usuario final. Si un producto las **satisface**, tiene calidad.
- Cada usuario tiene diferentes estándares de calidad

□ Ejemplo:

- **McDonald's** valora que la aplicación pueda realizar un pedido en tiempo y forma, **Burger** que puedan descargar descuentos, otros sólo con visualizar sucursales y otros con consultar precios están satisfechos.. etc

¿QUÉ ES CALIDAD?

- El **punto de vista del fabricante :**

- La define en términos de las **especificaciones originales** del producto. Si éste las cumple, tiene calidad.

- **Ejemplo:**

- Yo hago una aplicacion, defino la calidad de la aplicación en base a si cumple con las especificaciones que YO necesito



¿QUÉ ES CALIDAD?



- El **punto de vista del producto** :
 -
 - Tiene que ver con las características inherentes (funciones y características) de un producto
 - Cuanto de lo que se desea está presente en el producto?
 - El consumidor confunde “ **calidad** con **precio**”, entiende que mientras más caro es el producto, más cantidad de atributo está presente en el mismo.
 -

¿QUÉ ES CALIDAD?

- El **punto de vista basado en el valor** :
 - De acuerdo con lo que un cliente está dispuesto a **pagar** por un producto.
- Ejemplo:
 - McDonald's** pagará más dinero por una aplicación con mayor seguridad
 - Burger** pagará más dinero por un mejor diseño, usabilidad.

[illegible]

¿QUÉ ES CALIDAD?

- La **calidad del diseño** se refiere a las características que los diseñadores especifican para un producto (tipo de materiales, tolerancias y especificaciones del desempeño)

□

En el desarrollo del software, la *calidad del diseño* incluye el **grado en el que el diseño cumple** las funciones y características especificadas en el modelo de requerimientos.

□ Ejemplo:

- Macdonals requiere que se puedan visualizar las dif sucursales, los diseñadores implentan un módulo de visualización en google maps..

¿QUÉ ES CALIDAD?

La calidad es importante, pero que si el usuario no está satisfecho, nada de lo demás importa

¿QUÉ ES CALIDAD?

- Robert Glass [Ing de soft-escritor, especializado en calidad] afirma que es mejor plantear una relación más intuitiva:

satisfacción del usuario = producto que funciona + **buena calidad** + **entrega dentro del presupuesto y plazo**

CALIDAD DEL SOFTWARE

□ Cómo la definimos?

***Proceso eficaz de software** que se aplica de manera que crea un **producto útil** que proporciona valor medible a quienes lo **producen** y a quienes lo **utilizan**.*

□ Un proceso eficaz de software

- Infraestructura que da apoyo a cualquier esfuerzo de elaboración de un producto de software de alta calidad. (administración de proyectos, prácticas de Ing de soft, administración de los cambios, revisiones técnicas)

CALIDAD DEL SOFTWARE

▣ *Un producto útil*

- ▣ Entrega contenido, funciones y características que el **usuario final desea**;
- ▣ Satisface los requerimientos establecidos en forma explícita por los participantes.

▣ *Al agregar valor para el productor y para el usuario de un producto*

- ▣ Proporciona beneficios a la organización que lo produce y a la comunidad de usuarios finales.
- ▣ Software de alta calidad requiere un menor **esfuerzo de mantenimiento, menos errores** que corregir y **poca asistencia** al cliente.
- ▣ Usuarios obtienen valor agregado porque la aplicación provee una capacidad útil en forma tal que agiliza algún proceso de negocios.

Dimensiones de la calidad de Garvin

Sugiere que la calidad debe tomarse en cuenta, adoptando un punto de vista multidimensional que comience con la evaluación de la **conformidad** y termine con una visión trascendental (**estética**).

▣ Calidad del desempeño

- ▣ ¿El software entrega todo el contenido, las funciones y las características especificadas como parte del modelo de requerimientos, de manera que da valor al usuario final?

▣ Calidad de las características.

- ▣ ¿El software tiene características que sorprenden y agradan la primera vez que lo emplean los usuarios finales?

▣ Confiabilidad

- ▣ ¿El software proporciona todas las características y capacidades sin fallar?
- ▣ ¿Está disponible cuando se necesita? ¿Entrega funcionalidad libre de errores?

Dimensiones de la calidad de Garvin

□ Conformidad.

- ¿El software concuerda con los estándares locales y externos que son relevantes para la aplicación?
- ¿Concuerda con el diseño *de facto* y las convenciones de código?
- Por ejemplo, ¿la interfaz de usuario está de acuerdo con las reglas aceptadas del diseño para la selección de menú o para la entrada de datos?

□ Durabilidad.

- ¿El software puede recibir mantenimiento (cambiar) o corregirse (depurarse) sin la generación inadvertida de eventos colaterales?
- ¿Los cambios ocasionarán que la tasa de errores o la confiabilidad disminuyan con el tiempo?

Dimensiones de la calidad de Garvin

▣ Servicio

- ▣ ¿Existe la posibilidad de que el software reciba **mantenimiento** (cambios) o correcciones (depuración) a corto plazo?
- ▣ ¿El equipo de apoyo puede adquirir toda la información necesaria para hacer cambios o corregir defectos?

▣ Estética

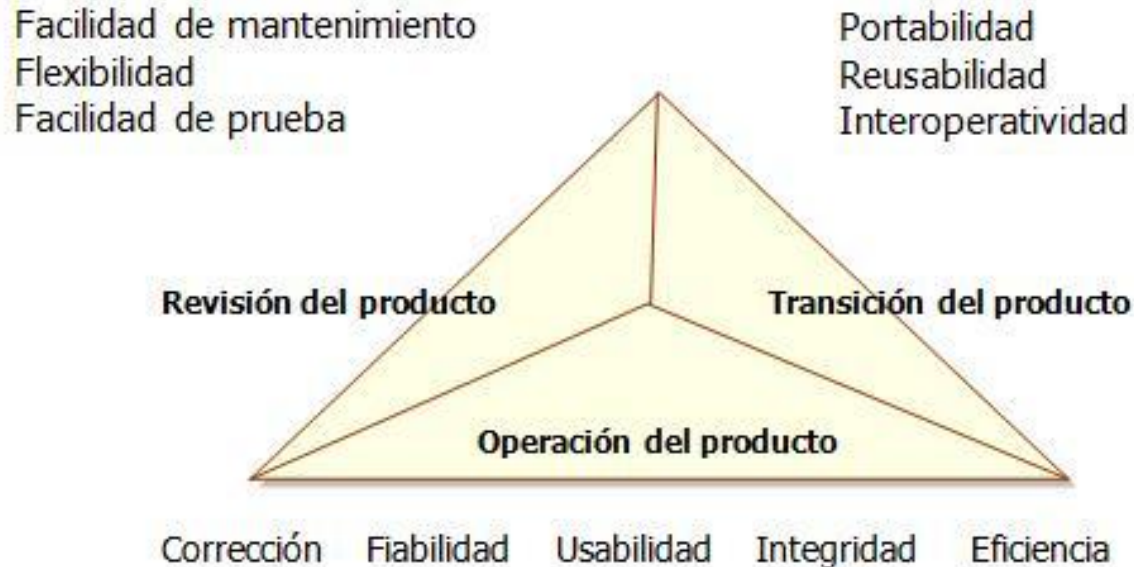
- ▣ Una entidad estética posee cierta **elegancia**, un flujo único y una “**presencia**” obvia que es difícil de cuantificar y que, no obstante, resulta evidente.

▣ Percepción.

- ▣ En ciertas situaciones, existen **prejuicios** que influirán en la **percepción** de la calidad por parte del usuario.
- ▣ **Por ejemplo**, si se introduce un producto de software elaborado por un proveedor que en el pasado ha demostrado mala calidad, la percepción de la calidad del producto tendrá influencia negativa. De manera similar, si un vendedor tiene una **reputación** excelente se percibirá buena calidad, aun si ésta en realidad no existe.

2.2 Factores de la calidad de McCall

Factores que afectan a la calidad del software



Factores de calidad de McCall (Fuente: Roger Pressman, 2005)

Factores de la calidad de McCall

OPERACIÓN DEL PRODUCTO: *Características operativas*

▣ **Corrección – Hace lo que quiero?**

- ▣ Grado en el que un programa satisface sus especificaciones y en el que cumple con los objetivos de la misión del cliente.

▣ **Confiabilidad – Lo hace de forma confiable todo el tiempo?**

- ▣ Grado en el que se espera que un programa cumpla con su función y con la precisión requerida

▣ **Eficiencia – Se ejecuta de la mejor manera?**

- ▣ Cantidad de recursos y de código requeridos por un programa para llevar a cabo su función.

Factores de la calidad de McCall

▣ **Integridad. – Es seguro?**

- ▣ Grado en el que es posible controlar el acceso de personas no autorizadas al software o a los datos.

▣ **Usabilidad – Esta diseñado para ser usado?**

- ▣ Esfuerzo que se requiere para aprender, operar, preparar las entradas e interpretar las salidas de un programa.

Factores de la calidad de McCall

REVISIÓN DEL PRODUCTO: *Capacidad de soportar cambios*

▣ **Facilidad de recibir mantenimiento.**

- ▣ Esfuerzo requerido para detectar y corregir un error en un programa

▣ **Flexibilidad.**

- ▣ Esfuerzo necesario para modificar un programa que ya opera.

▣ **Facilidad de prueba.**

- ▣ Esfuerzo que se requiere para probar un programa a fin de garantizar que realiza la función que se pretende.

Factores de la calidad de McCall

TRANSICIÓN DEL PRODUCTO: *Adaptabilidad a nuevos entornos*



▣ **Portabilidad.**

- ▣ Esfuerzo que se necesita para transferir el programa de un ambiente de sistema de hardware o software a otro.

Linux



▣ **Reusabilidad.**

- ▣ Grado en el que un programa (o partes de uno) pueden volverse a utilizar en otras aplicaciones (se relaciona con el empaque y el alcance de las funciones que lleva a cabo el programa).

▣ **Interoperabilidad.**

- ▣ Esfuerzo requerido para acoplar un sistema con otro.



Qué Factores de calidad que se persiguen?

Para hacer la evaluación, se necesita determinar atributos específicos y medibles

□ Intuitiva

- Grado en el que la **interfaz** sigue **patrones** esperados de uso, de modo que hasta un novato la pueda utilizar sin mucha capacitación.
 - ¿La **interfaz** lleva hacia una comprensión fácil?
 - ¿Todas las **operaciones** son fáciles de localizar e iniciar?
 - ¿La interfaz usa una metáfora reconocible?
 - ¿La **entrada** está especificada de modo que economiza el uso del teclado o del mouse?
 - ¿La **estética** ayuda a la comprensión y uso?
- Ejemplos de mala usabilidad:
 - <http://www.lingscars.com/>
 - <http://arngren.net/>
 - <http://dokimos.org/index.aspx>
 - http://www.web_4_all.republika.pl/



Qué Factores de calidad que se persiguen?

□ Eficiencia :

- Grado en el que es posible **localizar** o **iniciar** las **operaciones** y la **información**
 - ¿La **distribución** y **estilo** de la interfaz permite que un usuario introduzca con eficiencia las operaciones y la información?
 - ¿Una secuencia de operaciones (o entrada de datos) puede realizarse con economía de movimientos?
 - ¿Los **datos de salida** o el contenido están presentados de modo que se entienden de inmediato?
 - ¿Las operaciones jerárquicas están organizadas de manera que minimizan la profundidad con la que debe navegar el usuario para hacer que alguna se ejecute?



Ejemplos preceptos básicos de **navegabilidad**

- La facilidad de navegación dentro de un sitio es una de las razones por las cuales los usuarios deciden volver o no.
- **Menú de navegación**
 - permite al usuario desplazarse al menos por las principales secciones del mismo.



Ejemplos preceptos básicos de **navegabilidad**

❑ Evitar el empleo de botones “**volver**”

- ❑ En algunos sitios se puede ver que cuando se llega a ciertas secciones o páginas, la única alternativa que se presenta al usuario que se encuentra en ella es la de hacer click en un botón “**volver**” o “**atrás**”.
- ❑ Este tipo de navegación termina siendo desagradable para la mayoría de los usuarios, ya que para leer otro artículo debe seguir al menos dos pasos.

VOLVER



Ejemplos : preceptos básicos de **navegabilidad**

- **Volver en un click a la página principal**

- Incluir algún enlace a la [página de inicio](#) desde todas las demás páginas del sitio. Esto da seguridad al usuario y facilita el reinicio de la navegación por el sitio en caso de haberse extraviado.



Ejemplos : preceptos básicos de **navegabilidad**

□ Regla de los tres clicks

- Si se superan los **tres clicks**, es porque la navegabilidad necesita mejoras
- El paradigma de la navegabilidad es poder acceder desde cualquier parte del sitio a otro **con solo hacer un click**, aunque no siempre es posible.



Qué Factores de calidad que se persiguen?

▣ Robustez

- ▣ Grado en el que el software maneja **entradas erróneas** de datos o en el que se presenta interacción inapropiada por parte del usuario.
- ▣ ¿El software reconocerá el **error** si entran datos en el límite de lo permitido o más allá y, lo que es más importante, continuará operando sin fallar ni degradarse?
- ▣ ¿La interfaz reconocerá los errores cognitivos o de manipulación y guiará en forma explícita al usuario de vuelta al camino correcto?
- ▣ ¿La interfaz da un **diagnóstico y guía útiles** cuando se descubre una condición de error (asociada con la funcionalidad del software)?



Windows

A fatal exception 0E has occurred at 0028:C00068F8 in UxD UMM(01) + 000059F8. The current application will be terminated.

- * Press any key to terminate the application.
- * Press CTRL+ALT+DEL to restart your computer. You will lose any unsaved information in all applications.

Press any key to continue

Qué Factores de calidad que se persiguen?

▣ Riqueza :

- ▣ Grado en el que la interfaz provee un conjunto abundante de características.
- ▣ Puede **personalizarse** la interfaz según las necesidades específicas del usuario?
- ▣ ¿La interfaz tiene gran capacidad para permitir al usuario identificar una secuencia de operaciones comunes con una sola acción o comando?

Dur customizable system can meet
your specific business needs.



EL DILEMA DE LA CALIDAD DEL SOFTWARE

- Bertrand Meyer : “**punto medio mágico**” – Investigador, desarrollador

*Si produce un sistema de software de **mala calidad**, usted pierde porque nadie lo querrá comprar.*

*Por otro lado, si dedica un tiempo infinito, demasiado **esfuerzo y enormes** sumas de dinero para obtener un elemento perfecto de software, entonces tomará tanto tiempo terminarlo y será tan **caro** de producir ,que de todos modos quedará fuera del negocio.*

En cualquier caso, habrá perdido la ventana de mercado, o simplemente habrá agotado sus recursos.

*De modo que las personas de la industria tratan de situarse en ese **punto medio mágico** donde el producto es suficientemente bueno para no ser rechazado de inmediato, no en la evaluación, pero tampoco es un objeto perfeccionista ni con demasiado trabajo que lo convierta en algo que requiera demasiado tiempo o dinero para ser terminado.*

EL DILEMA DE LA CALIDAD DEL SOFTWARE

□ Software “suficientemente bueno”

Contiene las funciones y características de alta calidad que desean los usuarios, pero al mismo tiempo tiene otras más oscuras y especializadas que contienen errores conocidos.

El vendedor de software espera que la gran mayoría de usuarios finales **perdone los errores** gracias a que estén muy contentos con la funcionalidad de la aplicación.



EL DILEMA DE LA CALIDAD DEL SOFTWARE

▣ El costo de la calidad

- ▣ *La calidad es importante, pero cuesta tiempo y dinero —demasiado tiempo y dinero!!!*
- ▣ La calidad tiene un costo, pero la mala calidad también lo tiene —no sólo para los usuarios finales que deban vivir con el software defectuoso, sino también para la organización del software que lo elaboró y que debe darle mantenimiento



CAPITULO 2

Aseguramiento de la Calidad



Aseguramiento de la Calidad

El *aseguramiento de la calidad del software* (con frecuencia llamado **Administración de la calidad**) es una actividad sombrilla que se aplica en todo el proceso del software.

Incluye lo siguiente:

- 1) un **proceso** de **ACS**
- 2) **tareas específicas** de aseguramiento y **control de la calidad** (incluidas revisiones técnicas y una estrategia de pruebas relacionadas entre sí),
- 3) **prácticas eficaces de ingeniería de software** (métodos y herramientas),
- 4) **control** de todos los productos del trabajo de software y de los cambios que sufren
- 5) un **procedimiento** para garantizar el cumplimiento de los estándares del desarrollo de software (cuando sea aplicable)
- 6) mecanismos de **medición y reporte**.



Aseguramiento de la Calidad

- El grupo de **ACS** funciona como representante del cliente en el interior de la empresa.
- Ven al software desde el punto de vista del cliente.
- ¿El software cumple adecuadamente los factores de calidad mencionados?
- ¿El desarrollo del software se condujo de acuerdo con **estándares preestablecidos**?
- ¿Las **disciplinas técnicas** han cumplido con sus roles como parte de la actividad de ACS?

El **grupo de ACS** trata de responder éstas y otras preguntas para garantizar que se mantenga la calidad del software.

Control de la Calidad VS. Aseguramiento de la Calidad

CONTROL DE LA CALIDAD

- ❑ CC tiene como actividad principal el Testing.
- ❑ CC se relaciona con un producto o servicio específico.
- ❑ CC verifica si ciertos atributos se encuentran o no en un producto determinado.
- ❑ CC encuentra errores con el fin primordial de corregirlos.
- ❑ CC es responsabilidad del equipo.
- ❑ CC está referido a un producto.

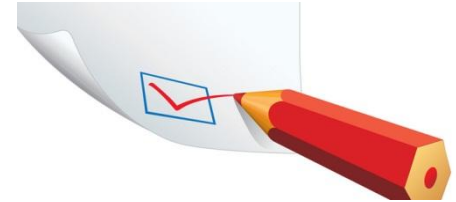
ASEGURAMIENTO DE LA CALIDAD

- ❑ AC ayuda a establecer Procesos.
- ❑ AC establece programas de medición para evaluar procesos.
- ❑ AC identifica debilidades en procesos y los mejora.
- ❑ AC es responsable de la administración de proyectos y procesos.
- ❑ AC se ocupa de todos los productos que serán en algún momento producidos por un proceso.
- ❑ AC está orientado a la Prevención
- ❑ AC Establece controles de calidad pero no desempeña el rol de la función del Control de Calidad como tal, inspeccionando productos.
- ❑ AC utiliza los resultados del Control de Calidad para evaluar y mejorar los “procesos” que generan los productos.

Un Poco de historia?

- Antes del siglo XX, el control de calidad era responsabilidad única del artesano que elaboraba el producto.
- Cuando pasó el tiempo , el control de calidad se convirtió en una actividad ejecutada por personas **diferentes** de aquellas que elaboraban el producto.
- La primera función formal de aseguramiento y control de la calidad se introdujo en los laboratorios **Bell en 1916** y se difundió con rapidez al resto del mundo de la manufactura.
- Durante la **década de 1940**, sugirieron enfoques más formales del control de calidad.

Cuáles son los Elementos de AC?



□ Estándares.

- El IEEE, ISO y otras organizaciones que establecen estándares han producido una amplia variedad de ellos para ingeniería de software y documentos relacionados.

- Los estándares los adopta de manera voluntaria una organización de software o los impone el cliente u otros participantes.

- El trabajo del ACS es asegurar que los estándares que se hayan adoptado **se sigan**, y que todos los productos del trabajo se **apeguen** a ellos.

□ Ejemplo:

□ ISO 9001:

- El estándar, que ha sido adoptado por más de 130 países para su uso, se está convirtiendo en el medio principal con el que los clientes pueden juzgar la competencia de un desarrollador de software.

□ Factores de calidad ISO 9126

- Funcionalidad
- Confiabilidad
- Usabilidad
- Eficiencia
- Facilidad de mantenimiento
- Portabilidad

Revisiones y auditorías.



- Las **revisiones técnicas** son una actividad del control de calidad que realizan ingenieros de software para otros ingenieros de software
- Las **auditorías** son un tipo de revisión efectuada por personal de ACS con objeto de garantizar que se sigan los lineamientos de calidad en el trabajo de la ingeniería de software.
- Por **ejemplo**,
 - una **auditoría** del proceso de revisión se efectúa para asegurar que las revisiones se lleven a cabo de manera que tengan la máxima probabilidad de descubrir errores
 - **Revisiones técnicas:** Los desarrolladores intercambian el código realizado con otros desarrolladores con el fin de obtener feedback.



Pruebas del Software

- Son una función del control de calidad que tiene un objetivo principal: **detectar errores**.
- El trabajo del AC es garantizar que las pruebas se **planeen** en forma apropiada y que se realicen con **eficiencia**, de modo que la probabilidad de que logren su objetivo principal sea máxima.
- **Ejemplo:**
 - AC revisa que los equipos de trabajo cuenten con las herramientas correctas para realizar las pruebas, de la forma en q se van a planear, documentar y reportar los errores.

Colección y análisis de los errores.

- La única manera de mejorar es medir cómo se está haciendo algo.
- El **ACS** reúne y analiza errores y datos acerca de los defectos para entender mejor cómo se cometen los errores y qué actividades de la ingeniería de software son más apropiadas para eliminarlos.



Administración del cambio.

- El cambio es uno de los aspectos que más irrumpe en cualquier proyecto de software.
- Si no se administra en forma adecuada, lleva a la confusión y ésta casi siempre genera mala calidad.
- El ACS asegura que se hayan instituido prácticas adecuadas de administración del cambio
- Ejemplo:
 - ACS controla que todo documento y código generado tenga versionado

Educación

- Toda organización de software quiere mejorar sus prácticas de ingeniería de software.
- Un contribuyente clave de la mejora es la **educación** de los ingenieros de software, de sus gerentes y de otros participantes.
- La organización de ACS es clave para proponer y patrocinar programas educativos.
- **Ejemplo:**
 - ACS organiza cursos de capacitación a los ingenieros de soft sobre «mejores prácticas de codificación»

Administración de la seguridad.

- Toda organización de software debe instituir políticas para proteger los datos en todos los niveles, establecer cortafuegos de protección para las *webapps* y asegurar que el software no va a ser vulnerado internamente.
- El **ACS** garantiza que para lograr la seguridad del software, se utilicen el proceso y la tecnología apropiados.



TAREAS, METAS Y MÉTRICAS DEL ACS

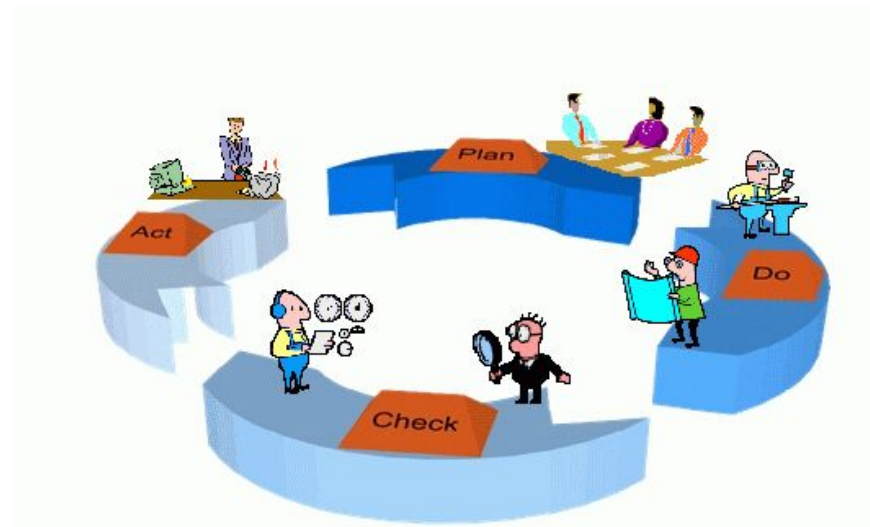
El aseguramiento de la calidad del software se compone de varias tareas asociadas con dos entidades diferentes:

- **Ingenieros de software** que hacen el trabajo técnico
- **Grupo de ACS** que tiene la responsabilidad de planear, supervisar, registrar, analizar y hacer reportes acerca de la calidad.



Tareas del ACS

- El objetivo del grupo de ACS es **auxiliar** al equipo del software para lograr un producto final de alta calidad.
- **Planeación, supervisión, registro, análisis y elaboración** de reportes para el aseguramiento de la calidad.



Tareas del ACS

1. Prepara el plan de ACS para un proyecto.

- El **plan** se desarrolla como parte de la preparación del proyecto y es revisado por todos los participantes.
- Las acciones de aseguramiento de la calidad efectuadas por el equipo de ingeniería de software y por el grupo de ACS son dirigidas por el plan.
- Éste identifica:
 - las **evaluaciones** que se van a realizar,
 - las **auditorías** y **revisiones** por efectuar,
 - los **estándares** aplicables al proyecto,
 - los **procedimientos** para reportar y dar seguimiento a los errores,
- Los productos del trabajo que genera el grupo de ACS y la retroalimentación que se dará al equipo del software.

CONTENIDO DEL PLAN DE ASEGURAMIENTO DE LA CALIDAD DEL SOFTWARE (PAQS)

1. Propósito

Describir el PAQS de la calidad para el proyecto X

2. Documento de referencia

Ver sección 4.2

3. Administración

3.1 Organización

Cada integrante es responsable de la calidad de su trabajo. Se tendrá un líder o Gestor de Calidad que tendrá el mando de los aspectos de calidad de todo el proyecto.

Para las actividades finales de pruebas (Pruebas de integración) se deberá asignar un equipo de ingenieros que incluya al líder o Gestor de Calidad.

3.2 Tareas

Las tareas deben incluir:

- Documentación
- Reuniones de revisión
- Verificaciones (incluye inspecciones)
- Actividades diseñadas para mejorar el proceso de AQ en sí

3.3 Responsabilidades

Líder o Gestor de Calidad: Es responsable de ver que se realicen las tareas de la sección 3.2 y que se siga lo prescrito en este documento, incluido el calendario de las revisiones especificadas.

Líder del Proyecto: será responsable de asegurar que la administración de la calidad se lleve a cabo.

Líder de Requerimientos: será responsable de ver que se realicen todas las inspecciones y revisiones al diseño preliminar y crítico.

4. Documentación

Se generarán los siguientes documentos:

- PAQS
- PACS
- PAPS

Tareas del ACS

2. Participa en el desarrollo de la descripción del software del proyecto.

- El equipo de software selecciona un proceso para el trabajo que se va a realizar.
- El grupo de ACS revisa la descripción del **proceso** a fin de cumplir con la política organizacional, los estándares internos para el software, los estándares impuestos desde el exterior (como la norma ISO-9001) y otras partes del plan del proyecto de software.

Tareas del ACS

3. Revisa las actividades de la ingeniería de software a fin de verificar el cumplimiento mediante el proceso definido para el software.

El grupo de ACS identifica, documenta y da seguimiento a las desviaciones del proceso y verifica que se hayan hecho las correcciones pertinentes.

4. Audita los productos del trabajo de software designados para verificar que se cumpla con aquellos definidos como parte del proceso de software.

- El grupo de ACS revisa productos del trabajo seleccionados; identifica, documenta y da seguimiento a las desviaciones; verifica que se hayan hecho las correcciones necesarias y reporta periódicamente los resultados de su trabajo al gerente del proyecto.

Tareas del ACS

5. Asegura que las desviaciones en el trabajo de software y sus productos se documenten y manejen de acuerdo con un **procedimiento** documentado.

- Las **desviaciones** pueden encontrarse en el plan del proyecto, la descripción del proceso, los estándares aplicables o los productos del trabajo de la ingeniería de software

6. Registra toda falta de cumplimiento y la reporta a la alta dirección.

- Se da seguimiento a los **incumplimientos** hasta que son resueltos.

Metas, atributos y métricas del ACS

▣ Calidad de los requerimientos.

- ▣ La **corrección, completitud y consistencia** del modelo de requerimientos tendrá una gran influencia en la calidad de todos los productos del trabajo que sigan.
- ▣ El ACS debe garantizar que el equipo de software ha **revisado** en forma apropiada el modelo de requerimientos a fin de alcanzar un alto nivel de calidad.

▣ Calidad del diseño.

- ▣ Todo elemento del modelo del diseño debe ser evaluado por el equipo del software para asegurar que tenga alta calidad y que el diseño en sí **se apegue a los requerimientos**.
- ▣ El ACS busca atributos del diseño que sean indicadores de la calidad.

Metas, atributos y métricas del ACS

▣ Calidad del código.

- ▣ El código fuente y los productos del trabajo relacionados (por ejemplo, otra información descriptiva) deben **apegarse a los estándares locales de codificación** y tener características que faciliten darle mantenimiento.
- ▣ El ACS debe identificar aquellos atributos que permitan hacer un análisis razonable de la calidad del código.

▣ Eficacia del control de calidad.

- ▣ Un equipo de software debe aplicar recursos limitados, en forma tal que tenga la máxima probabilidad de lograr un resultado de alta calidad.
- ▣ ACS analiza la **asignación de recursos para las revisiones y pruebas** a fin de evaluar si se asignan en la forma más eficaz.

FIGURA 16.1**Metas atributos y métricas de la calidad del software***Fuente: Adaptado de [Hya96].*

Meta	Atributo	Métrica
Calidad de los requerimientos	Ambigüedad	Número de modificadores ambiguos (por ejemplo, muchos, grande, amigable, etc.)
	Compleitud	Número de TBA y TBD
	Comprensibilidad	Número de secciones y subsecciones
	Volatilidad	Número de cambios por requerimiento Tiempo (por actividad) cuando se solicita un cambio
	Trazabilidad	Número de requerimientos no trazables hasta el diseño o código
	Claridad del modelo	Número de modelos UML Número de páginas descriptivas por modelo Número de errores de UML
Calidad del diseño	Integridad arquitectónica	Existencia del modelo arquitectónico
	Compleitud de componentes	Número de componentes que se siguen hasta el modelo arquitectónico Complejidad del diseño del procedimiento
	Complejidad de la interfaz	Número promedio de pasos para llegar a una función o contenido normal Distribución apropiada
	Patrones	Número de patrones utilizados
Calidad del código	Complejidad	Complejidad ciclomática
	Facilidad de mantenimiento	Factores de diseño (capítulo 8)
	Comprensibilidad	Porcentaje de comentarios internos Convenciones variables de nomenclatura
	Reusabilidad	Porcentaje de componentes reutilizados
	Documentación	Índice de legibilidad

**Eficacia del control
de calidad**

Asignación de recursos

Tasa de finalización

Eficacia de la revisión

Eficacia de las pruebas

Porcentaje de personal por hora y por actividad

Tiempo de terminación real versus lo planeado

Ver medición de la revisión (capítulo 14)

Número de errores de importancia crítica encontrados

Esfuerzo requerido para corregir un error

Origen del error

La norma ISO 9001:2000

- A fin de que una organización de software se registre en la ISO 9001:2000, debe establecer **políticas y procedimientos** que cumplan cada uno de los requerimientos y después demostrar que sigue dichas políticas y procedimientos.
- Para registrarse en alguno de los modelos del sistema de aseguramiento de la calidad contenidos en la ISO 9000, por medio de **auditores externos** se revisan en detalle el sistema y las operaciones de calidad de una compañía, respecto del cumplimiento del estándar y de la operación eficaz.
- Después de un registro exitoso, el grupo de registro representado por los **auditores** emite un **certificado** para la compañía.
- Auditorías semestrales de supervisión aseguran el cumplimiento continuo de la norma.

Elementos básicos ISO 9001:2000



- Establecer los elementos de un sistema de administración de la calidad.
 - Desarrollar, implementar y mejorar el sistema.
 - Definir una política que ponga el énfasis en la importancia del sistema.
- Documentar el sistema de calidad.
 - Describir el proceso.
 - Producir un manual de operación.
 - Desarrollar métodos para controlar (actualizar) documentos.
 - Establecer métodos de registro.
- Apoyar el control y aseguramiento de la calidad.
 - Promover la importancia de la calidad entre todos los participantes.
 - Centrarse en la satisfacción del cliente.
 - Definir un plan de calidad que se aboque a los objetivos, responsabilidades y autoridad.
 - Definir mecanismos de comunicación entre los participantes.

Elementos básicos ISO 9001:2000

- Establecer **mecanismos de revisión** para el sistema de administración de la calidad.
 - Identificar métodos de revisión y mecanismos de retroalimentación.
 - Definir procedimientos para dar seguimiento.
- **Identificar recursos para la calidad**, incluidos personal, capacitación y elementos de la infraestructura.
 - Establecer mecanismos de control.
 - Para la planeación
 - Para los requerimientos del cliente
 - Para las actividades técnicas (tales como análisis, diseño y pruebas)
 - Para la 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.

Capitulo 3

Qué es el Testing?

Capítulo 3

Qué es el Proceso de Testing?

Capitulo 3

Qué son Pruebas de Software ?



Client

Requirements

Reverse
engineer

Use Case

Documents

User Stories

QA Team

Test Case

Data Set

Scenarios

Environment
s

traceability

Traceability

Bugs/Issue

Evidence

Reports

Metrics

Work
Products



Qué son las Pruebas?

- ▣ **Verificación** del comportamiento de un programa en un conjunto finito de casos de prueba, debidamente seleccionados de por lo general infinitas ejecuciones de dominio, contra la del comportamiento esperado.
- ▣ Son una serie de **actividades** que se realizan con el propósito de encontrar los posibles **fallos** de implementación, calidad o usabilidad de un programa; probando el comportamiento del mismo.

Qué son las Pruebas?

- La prueba es un **conjunto de actividades** que pueden planearse por adelantado y realizarse de manera sistemática.
- Durante el proceso de software, debe definirse una **plantilla** para la prueba del software: un conjunto de pasos que incluyen métodos de prueba y técnicas de diseño de casos de prueba específicos.
- El proceso de recopilación de información mediante observaciones y compararlas con las expectativas.



Qué son las Pruebas?

- Es un proceso de ejecución de un programa con la intención de encontrar errores.
- Es una investigación técnica y empírica de un producto, hecho en nombre de los interesados, con la intención de revelar información relacionada con la calidad de un producto o servicio.
- **Según IEEE:**
 - El proceso de funcionamiento de un sistema o componente bajo ciertas condiciones, observar o registrar los resultados, y hacer una evaluación de algún aspecto del sistema o componente.

- Para realizar una prueba efectiva, debe **realizar revisiones técnicas efectivas** para eliminar muchos errores antes de comenzar la prueba.
- Diferentes **técnicas de prueba** son adecuadas para distintos enfoques de ingeniería de software y en diferentes momentos en el tiempo.
- **Prueba y depuración** son actividades diferentes, pero la depuración debe incluirse en cualquier estrategia de prueba.
- Una estrategia para la prueba de software debe incluir **pruebas de bajo nivel**, que son necesarias para verificar que un pequeño segmento de código fuente se implementó correctamente, así como **pruebas de alto nivel**, que validan las principales funciones del sistema a partir de los requerimientos del cliente.
- Una **estrategia** debe proporcionar una **guía** para el profesional y un conjunto de guías para el jefe de proyecto.

Objetivos de las pruebas

- Detección de **errores**
- Generación de **Confianza** respecto al nivel de calidad
- **Confirmación de la funcionalidad:**
 - La información del sistema debe ser implementada tal y cómo ha sido especificada
- **Aporte** de Información para la toma de decisiones
- **Prevención** de defectos
- Un buen caso de prueba es el que tiene alta probabilidad de encontrar un error que no se ha detectado hasta el momento.
- Una prueba exitosa es la que **descubre** un error no detectado hasta el momento.

Cuántas pruebas son suficientes?

- No encontrar más defectos es un criterio apropiado para finalizar las actividades de pruebas.
- Son necesarias otras métricas para reflejar de forma adecuada el nivel de calidad alcanzado
 - **Pruebas basadas en el riesgo**
 - Determina el grado en el cual se ha probado, es decir la responsabilidad en caso de fallos, Probabilidad de la ocurrencia de fallos, aspectos relativos a factores económicos y propios del proyecto
 - **Pruebas basadas en Plazos y Presupuesto**
 - La disponibilidad de los recursos: (Personal, tiempo, presupuesto), determinan la medida del esfuerzo en el proceso de Pruebas.

Qué es un caso de prueba?

- ❑ Conjunto de **condiciones o variables** bajo las cuáles el analista determinará si el requisito de una aplicación es parcial o completamente satisfactorio.
- ❑ Se pueden realizar muchos casos de prueba para determinar que un **requisito** es completamente satisfactorio.
- ❑ Con el propósito de comprobar que todos los requisitos de una aplicación son revisados, debe haber **al menos un caso de prueba** para cada requisito a menos que un requisito tenga requisitos secundarios. En ese caso, cada requisito secundario deberá tener por lo menos un caso de prueba.
- ❑ Contienen una **entrada conocida** y una **salida esperada**, los cuales son formulados antes de que se ejecute la prueba. La *entrada conocida* debe probar una precondition y la *salida esperada* debe probar una postcondición.
- ❑ Incluyen una descripción de la **funcionalidad** que se probará, la cuál es tomada ya sea de los requisitos o de los casos de uso.



Estructura de los casos de prueba

- ▣ **Introducción/visión general:** Contiene información general acerca de los Casos de Prueba.
- ▣ **Identificador:** Es un identificador único para futuras referencias, por ejemplo, mientras se describe un defecto encontrado.
- ▣ **Caso de prueba dueño/creador:** Es el nombre del analista o diseñador de pruebas, quien ha desarrollado pruebas o es responsable de su desarrollo.
- ▣ **Versión:** La actual definición del caso de prueba.
- ▣ **Nombre:** El caso de prueba debe ser un título entendible por personas, para la fácil comprensión del propósito del caso de prueba y su campo de aplicación.
- ▣ Identificador de **requerimientos** el cuál está incluido por el caso de prueba. También aquí puede ser un identificador de casos de uso o de especificación funcional.
- ▣ **Propósito:** Contiene una breve descripción del propósito de la prueba, y la funcionalidad que chequea.
- ▣ **Dependencias:** Indica qué otros subsistemas están involucrados y en qué grado.

Estructura de los casos de prueba

- ▣ **Actividades** de los casos de prueba
 - ▣ **Ambiente de prueba/configuración:** Contiene información acerca de la configuración del hardware o software en el cuál se ejecutará el caso de prueba.
 - ▣ **Inicialización:** Describe acciones, que deben ser ejecutadas antes de que los casos de prueba se hayan inicializado. Por ejemplo, debemos abrir algún archivo.
 - ▣ **Finalización:** Describe acciones, que deben ser ejecutadas después de realizado el caso de prueba.
 - ▣ Por ejemplo si el caso de prueba estropea la base de datos, el analista debe restaurarla antes de que otro caso de prueba sea ejecutado.
 - ▣ **Acciones:** Pasos a realizar para completar la prueba.
 - ▣ Descripción de los **datos de entrada**

Estructura de los casos de prueba

■ Resultados

- **Salida esperada:** Contiene una descripción de lo que el analista debería ver tras haber completado todos los pasos de la prueba.
- **Salida obtenida:** Contiene una breve descripción de lo que el analista encuentra después de que los pasos de prueba se hayan completado.
- **Resultado:** Indica el resultado cualitativo de la ejecución del caso de prueba, a menudo con un **Correcto/Fallido**.
- **Severidad:** Indica el impacto del defecto en el sistema: Grave, Mayor, Normal, Menor.
- **Evidencia:** En los casos que aplica, contiene un link al print de pantalla (screenshot) donde se evidencia la salida obtenida.
- **Seguimiento:** Si un caso de prueba falla, frecuentemente la referencia al defecto implicado se debe enumerar en esta columna. Contiene el código correlativo del defecto, a menudo corresponde al código del sistema de tracking de bugs que se esté usando.
- **Estado:** Indica si el caso de prueba está: No iniciado, En curso, o terminado.

Caso de Prueba

Caso de Prueba	
ID	TC001
Título	Login
Precondiciones	Usuario debería estar registrado en el sistema
Descripción	Verificar funcionamiento Login utilizando datos válidos
Prioridad	Alta/Media/Baja
Pasos:	<ol style="list-style-type: none">1. Ingresar usuario válido2. Ingresar contraseña válida3. Click Botón "Entrar"4. Verificar Nombre usuario esquina superior derecha5. Verificar que se muestre la opción "salir"
Resultado Esperado:	El usuario debería *loguearse* en el sistema y el sistema debería mostrar mensaje de Bienvenida. El nombre del usuario debería visualizarse en la esquina superior derecha con una opción de "salir"
Ambiente:	Version sistema: 1.2.3 Firefox, Opera, Mac OSX, Windows 7
Estado:	No ejecutado/Bloqueado/Pasó/Falló
Corrida:	1
Creado por:	Julieta Barrionuevo
Ejecutado por:	Julieta Barrionuevo

ID	Título	Precondiciones	Descripción	Prioridad	Pasos:	Resultado Esperado:	Ambiente:	Estado:	Corrida:	Creado por:	Ejecutado por:
TC001	Login Positivo	Usuario debería estar registrado en el sistema	Verificar funcionamiento Login utilizando datos válidos	Alta	1. Ingresar usuario válido 2. Ingresar contraseña válida 3. Click Botón "Entrar" 4. Verificar Nombre usuario esquina superior derecha 5. Verificar que se muestre la opción "salir"	El usuario debería *loguearse* en el sistema y el sistema debería mostrar mensaje de Bienvenida. El nombre del usuario debería visualizarse en la esquina superior derecha con una opción de "salir"	Version sistema: 1.2.3 Firefox, Opera, Mac OSX, Windows 7	Pasó	1	Julietta Barrionuevo	Julietta Barrionuevo
TC002	Login Negativo	Usuario debería estar registrado en el sistema	Verificar funcionamiento Login utilizando datos inválidos	Alta	1. Ingresar usuario inválido 2. Ingresar contraseña válida 3. Click Botón "Entrar"	El sistema debería mostrar un mensaje de error indicando al usuario que los datos ingresados fueron incorrectos	Version sistema: 1.2.3 Firefox, Opera, Mac OSX, Windows 7	Falló	1	Julietta Barrionuevo	Julietta Barrionuevo

Consejos..

- Cada modulo del sistema debe tener al menos **un caso** de prueba.
- Analizar **datos de prueba**, datos necesarios para correr las pruebas
- Comenzar con casos de prueba **positivos** (satisface requerimientos) y luego plantear los casos **negativos**.
- Deben ser **claros, breves y simples**, un test case una funcionalidad.
- Considerar **clonar, mantener, separar, conectar y mantener su trazabilidad** (requerimiento/reglas de negocio, con el test case y luego el bug). La Trazabilidad es bidireccional.
- Cuando es “**forzado**” el curso normal de un sistema se debe manejar caminos alternativos. Si se pasa por alto el manejo de excepciones esto puede garantizar la existencia de FALLAS.
- Hacer los **test cases** de forma que se puedan **reutilizar** o **clonar**.
- Considerar los “**actores**” o “**usuarios**” del sistema

Consejos..

- ❑ Considerar los diferentes “**escenarios**”
- ❑ Considerar el **impacto** que va a tener la corrida del test case en el resto del sistema y la BD
- ❑ Probar siempre primero lo **obvio** (positivo y negativo)
- ❑ Analizar la “**volatilidad**” del test case y de los datos (mantenimiento, general, específico)
- ❑ Mantener un **versionado** de las pruebas en base a las diferentes versiones del sistema.
- ❑ Tratar de mantener la **trazabilidad** ante mantenimientos o nuevas iteraciones: Requerimientos /caso de prueba/error.
- ❑ Analizar la cobertura del sistema al terminar de armar las pruebas, tratando de cubrir lo máximo dentro de los límites, todos los caminos, subflujos, alternativas.

Axiomas de Myers

- Un buen conjunto de datos de prueba es el que posee una gran probabilidad de detectar un error no descubierto, no aquel que muestra que el programa se comporta correctamente.
- Uno de los problemas más difíciles con que se tropieza en una prueba es saber cuándo **detenerse**.
- Es imposible que usted pruebe su **propio** programa.
- La descripción de la información de salida o de los resultados esperados, es un elemento imprescindible de todo conjunto de datos de prueba.
- Evite las pruebas no repetibles o reproducibles.
- Desarrolle **datos de prueba** que contengan información de entrada relativa a condiciones válidas o inválidas.
- **Examine y revise** cuidadosamente los resultados de cada prueba.
- Con el incremento del número de **errores** encontrados en un programa, aumenta igualmente la probabilidad de la existencia de errores no descubiertos.
- Asigne la tarea de prueba a los integrantes de un equipo con mayor **creatividad**.
- Asegúrese que se hallan contemplado en el **diseño y estructura** del programa, las facilidades para una prueba adecuada.
- El diseño del sistema debería contemplar y asegurar que cada módulo sea integrado con el sistema una sola vez.
- No altere nunca el programa para que la prueba resulte más fácil.
- La prueba, como cualquier otra actividad, debe comenzar con el establecimiento de los **objetivos** pertinentes

Probar significa más que ejecutar Pruebas!

- La **ejecución de las pruebas** es sólo una parte de las pruebas
- El **proceso de prueba** incluye:
 - Planificación y control
 - Selección de condiciones de Prueba
 - Comprobación de Resultados
 - Generación de Informes respecto del proceso de pruebas y el sistema sujeto a pruebas
 - Finalizar y completar actividades de cierre.
- La revisión de documentos, código fuente y la realización de análisis estático también ayudan a prevenir la aparición de defectos en el código.

7 Principios del proceso de Pruebas

Cuando se parar una corrida de un test case??

- Cuando se encuentra un error grave (por ej un major, crash o block). Considerar cuando existe requerimiento de mayor prioridad
- Cuando cada vez hay más errores, considerar en qué punto se debe parar.
- Analizar el impacto de lo que se va a probar
- Cuando no se puede acceder al sistema.
- Cuando se cae la base de datos
- Cuando no hay ambiente seguro (considerar si uno está en producción o en un ambiente controlado, y a su vez con la versión/delivery correcto)
- Cuando no se tiene una BD de prueba para poder verificar errores. Analizar el DD, interfaz con salidas, layers, etc. Que no sea impactada la BD de producción, que por lo menos sea la de desarrollo, puede estar llena de basura. Crear una copia de una BD limpia para ser reutilizada. Se puede pedir una BD reducida de la BD que está en producción para tener datos reales o semejantes.
- Cuando agotaste todos los caminos y no se hallan más errores

□ Principio 1 – El Testing demuestra la presencia de errores.

- Mediante el testing podemos demostrar la presencia de **errores**, pero no la ausencia de los mismos. Incluso si no se detectan deficiencias, no es una prueba de la corrección.
- Que muchas pruebas se hayan ejecutado sin encontrar un error, no demuestra que “No hay errores”.
- Tan pronto como nos encontramos con un error, se ha mostrado ‘Este código no está libre de errores’

□ Principio 2 – El testing exhaustivo es imposible

- Probar todas las combinaciones de entradas y condiciones es imposible (salvo en casos triviales).
- En vez de intentar probar todo, debemos enfocar las pruebas en base a riesgos y prioridades:
 - qué testear primero
 - qué testear más
 - qué tan profundo testear cada ítem
 - qué no testear (por esta vez)
- El cliente y los managers van a querer realizar una cantidad de testing que les provea un retorno de la inversión (ROI).

□ Principio 3 – Testing Temprano

- Las actividades de prueba deben comenzar tan pronto como sea posible en el software o el sistema.
- Las pruebas (Test cases) deben enfocarse en los objetivos definidos en el Test Plan.
- El objetivo principal de las pruebas estáticas es llevar a cabo las pruebas tan pronto como sea posible, encontrar y corregir defectos de forma más barata y prevenir los defectos que aparezcan en etapas posteriores de este proyecto.
- Estas actividades nos ayudan a conocer al principio los defectos e identificar los posibles grupos. Hay que recordar que a partir de la definición de las pruebas, las pruebas no se inician hasta que el código ha sido terminado.

□ Principio 4 – Agrupamiento de defectos

- Un pequeño número de módulos contienen la mayoría de los defectos detectados durante las pruebas de prelanzamiento.
- Un fenómeno que los testers han observado es que muchos defectos tienden a agruparse. Esto puede suceder debido a un área del código es particularmente compleja y delicada, o porque el cambio de software y otros productos tiende a causar una reacción en cadena.
- Igualmente, hay que tener en cuenta que este agrupamiento de defectos va cambiando en el tiempo, por eso es importante centrarse en los “puntos calientes”.
- Los testers suelen utilizar esta información al hacer su evaluación de riesgos para la planificación de las pruebas, y se centrarán en los conocidos “puntos calientes”.

□ Principio 5 – La paradoja del pesticida

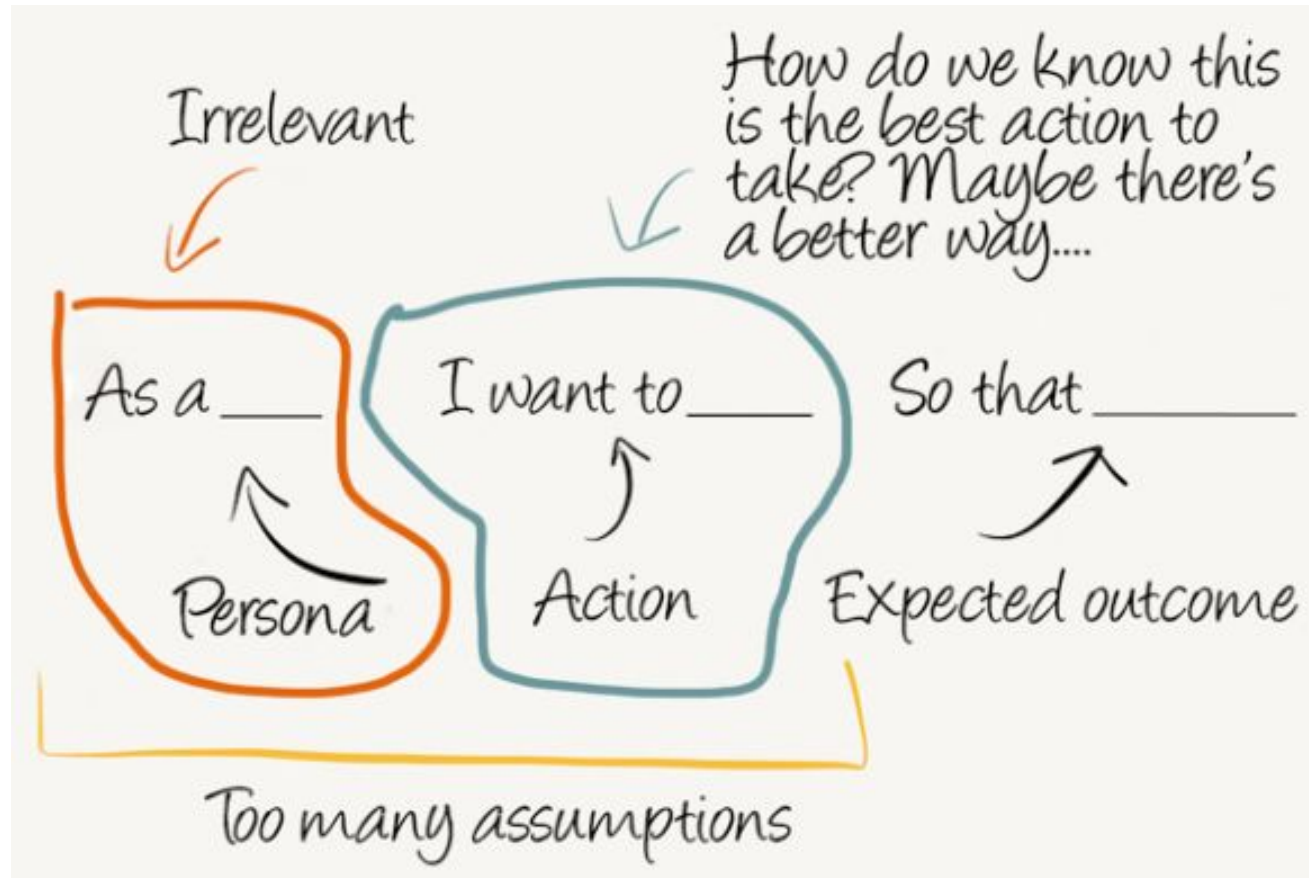
- Si las pruebas se repiten una y otra vez. Con el tiempo el mismo conjunto de casos de prueba ya no encuentran nuevos errores.
- Todas las técnicas de prueba se dirigen a un conjunto diferente de bugs.
- Si los programadores responden a las pruebas y la información de los errores mediante la reducción o eliminación de tales errores, se deduce que como su software mejora, la eficacia de las pruebas anteriores se daña, es decir, las pruebas se desgastan y tendrán que aprender, crear y utilizar nuevas pruebas basadas en las nuevas técnicas para captar nuevos errores.
- Para superar esta "**paradoja de pesticidas**", los test cases deben ser examinados y revisados periódicamente.
- Pruebas nuevas y diferentes deben ser escritas para ejercitar las distintas partes del software o el sistema para encontrar potencialmente más defectos. Con el tiempo, nuestro enfoque puede cambiar de encontrar errores de codificación, a mirar los requisitos y documentos de diseño, y en busca de mejoras en los procesos para que podamos prevenir los defectos en el producto.

- **Principio 6** – El testing es dependiente del contexto
- Las pruebas se realizan de manera diferente en diferentes contextos.
- **Por ejemplo,**
 - la seguridad del software será testeada de forma diferente en un sitio de comercio electrónico que uno donde se comparten fotografías.
- No todos los sistemas de software llevan el mismo nivel de riesgo y no todos los problemas tienen el mismo impacto cuando se producen.
- Algunos de los problemas que encontramos en el uso de software son bastante triviales, pero otros pueden ser costosos y perjudiciales - provocando pérdida de reputación, de dinero, tiempo o de negocios - e incluso puede resultar en lesiones o la muerte.

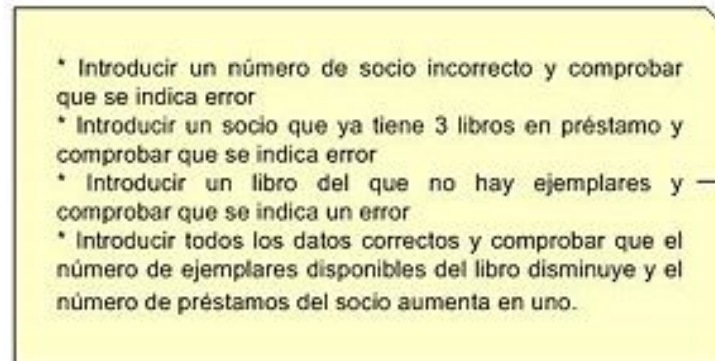
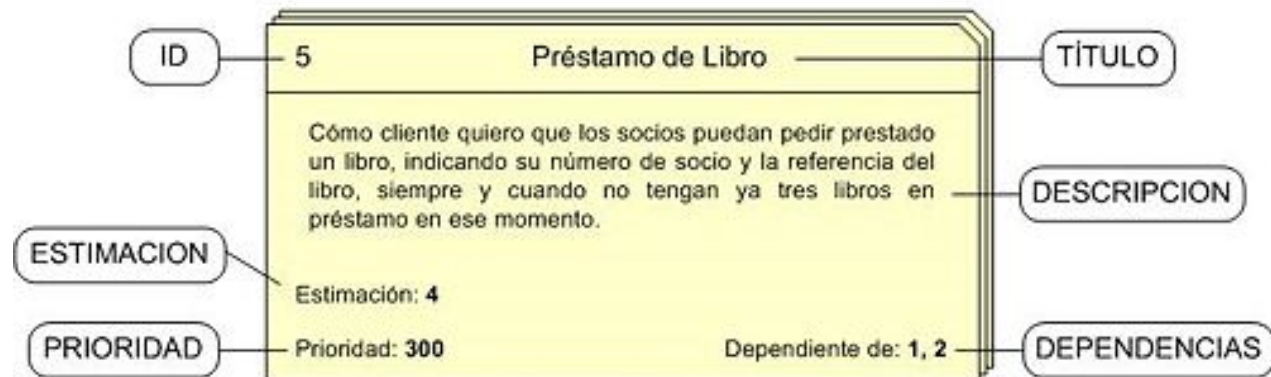
□ Principio 7 – Falacia sobre la ausencia de errores

- Encontrar y corregir defectos no sirve de nada si el sistema integrado no se puede utilizar y no satisfacer las necesidades de los usuarios y sus expectativas.
- Las personas y organizaciones que compran y utilizan software como ayuda en su día a día no están interesadas en los defectos o el número de defectos, salvo que sean directamente afectados por la inestabilidad del software.
- La gente que usa software está más interesada en que la aplicación los apoye en la realización de sus tareas de manera eficiente y eficaz.
- Por eso las revisiones en las **primeras etapas (requisitos y diseño)** son una parte importante de las pruebas y si los verdaderos usuarios del sistema no han estado involucrados en el proceso de prueba en algún momento, entonces es probable que no obtengan lo que realmente quieren, por ejemplo las páginas web puede parecer libre de errores, pero si han sido diseñados de una manera que no es compatible con los procesos de negocio del sistema no se podrán utilizar

User stories



ANVERSO



REVERSO

Contras de no armar test cases

- Probar sin definir previamente los casos, podría ocasionar que:
 - Queden funcionalidades **sin probar**.
 - Se complique la definición de **tiempos** y **prioridades**.
 - Se dificulte la generación de **datos previa**.
 - Se pruebe varias veces lo mismo.
 - Sea difícil determinar cuándo **parar** de testear
 - Sea difícil detectar si se llegaron a cubrir o no las funcionalidades críticas.
 - Si hubiera que recortar el tiempo de test, no se dispondría de información para hacerlo.

Conjunto de Pruebas (Test Suite)

- **Conjunto de casos de pruebas** para pruebas manuales de sistema. Es solo una actividad en el plan de QA.
- Un conjunto de casos de prueba es simplemente una tabla de contenidos para casos de prueba individuales.
- Organizando el conjunto de casos de prueba por prioridad, área de funcionalidad, objeto de negocio o versión puede ayudar a identificar partes del sistema que necesitan pruebas adicionales.

TC cortos vs. TC Largos

□ TC Cortos

□ Ventajas:

- Se crean en poco tiempo
- Son fáciles de editar y mantener

□ Desventajas

- Difíciles de comprender por otra persona

TC cortos vs. TC Largos

□ TC Largos

□ Ventajas:

- Brindan mucha información a la persona encargada de ejecutarlo

□ Desventaja:

- Se demora mucho tiempo en crearlos
- Son laboriosos a la hora de editarlos

Revisión x Pares (Ejemplo)

#	Items	Descripción	Pasó
1	Asociación al proyecto	El test case debe estar asociado al proyecto	SI
2	Asociación a un Módulo	Los test cases deben asociarse a un Módulo	SI
3	ID / Nombre único	El test case debe tener un nombre descriptivo y un ID unico	SI
4	Descripción	El Test case debe tener una descripción de lo que se pretende testear	No
5	Autor	El test case debe tener siempre un autor	SI
7	Pasos	Los pasos deben ser descriptivos	SI
8	Consultas	En caso de que haya interacción con la base de datos, insertar la consulta que realiza.	N/A
9	Resultado esperado	Los pasos deben tener un resultado esperado de lo que debería hacer el sistema	SI
			85,00%

Proceso del Testing

1. 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 las pruebas que se pretende abordar.
- Esto nos dará lo que comúnmente se conoce como la **misión** de las pruebas o la asignación de las pruebas.
- Basado en este entendimiento, que establece las metas y objetivos de la prueba en sí, podemos obtener un enfoque de las pruebas.
- El **Control** de las pruebas es la actividad permanente de comparar el progreso real contra el plan, y comunicar el estado actual de las pruebas, incluyendo las desviaciones del plan.
- Se trata de tomar las **medidas** necesarias para cumplir la misión y los objetivos del proyecto.

Proceso del Testing

2. Análisis y Diseño

- Tiene como tareas principales la revisión de la base de pruebas ,tales como :
 - Los requerimientos del producto,
 - La **arquitectura**,
 - Las especificaciones de diseño e interfaces,
 - La verificación de las **especificaciones** para el software bajo pruebas;
 - Evaluar la **testeabilidad** de los requisitos y el sistema;
 - **Identificar y priorizar** las condiciones de prueba;
 - Identificar los **datos** necesarios de prueba;
 - **Diseño y priorización** de los casos de las pruebas ;
 - Diseño del entorno de prueba e identificación de cualquier infraestructura necesaria y las herramientas.

Proceso del Testing

3. Aplicación y Ejecución

□ La aplicación de las pruebas tiene las siguientes tareas principales:

- **Desarrollar** y dar **prioridad** a nuestros **casos de prueba**.
- También se escriben las **instrucciones** para la realización de las pruebas (procedimientos de prueba), así como crear los datos de prueba para ellos. Opcionalmente, es posible que necesitemos automatizar algunas pruebas utilizando arneses de prueba y scripts de prueba automatizados.
- Creación de **conjuntos de pruebas** de los casos de prueba para la ejecución de la prueba (Test Suits). Un conjunto de pruebas es una colección lógica de casos de prueba que, naturalmente, trabajan juntos.
Las Test suits a menudo comparten datos y un alto nivel común de un conjunto de objetivos. También vamos a establecer un calendario de ejecución de las pruebas.
- Implementar y verificar el **ambiente**. Nos aseguramos de que el entorno de pruebas se ha configurado correctamente, posiblemente, incluso la realización de pruebas específicas sobre el mismo.

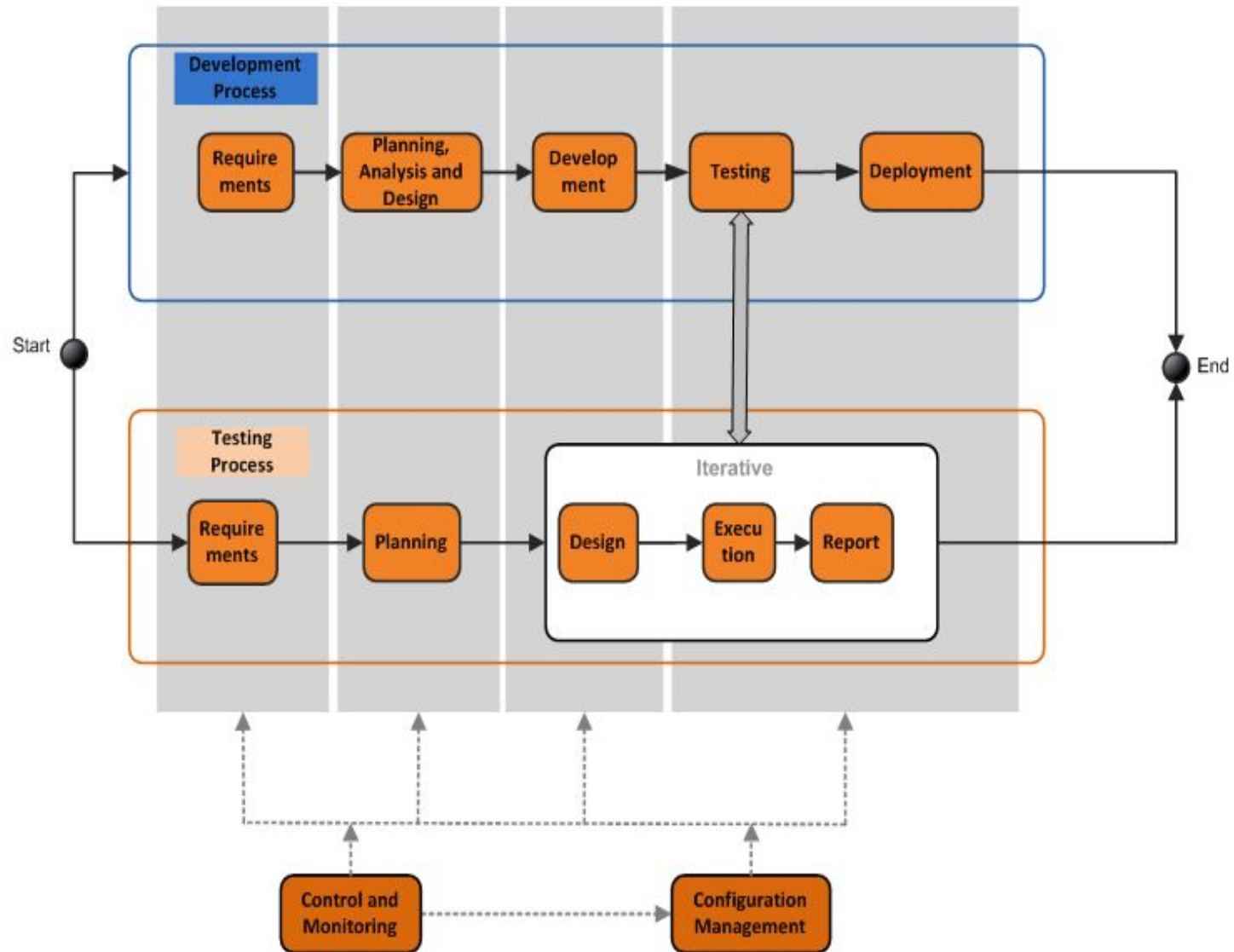
□ Por otro lado, **ejecución de las pruebas** tiene las siguientes tareas principales:

- **Ejecutar el set de pruebas**, siguiendo nuestros procedimientos de pruebas. Podemos hacerlo manualmente o mediante el uso de herramientas de prueba de ejecución, de acuerdo con la secuencia prevista, la ejecución de la mayoría más importantes en primer lugar.
- **Registrar el resultado** de la ejecución de pruebas y registrar la identidad y las versiones del software en las herramientas de pruebas.
- Debemos saber exactamente lo que en las pruebas se utilizó contra la versión del software, tenemos que informar defectos con **versiones** específicas, y el registro de las pruebas para un registro de auditoría.
- Comparar los **resultados reales** (lo que pasó cuando nos encontramos con las pruebas) con los resultados **esperados** (lo que habíamos anticipado que ocurriría).

Proceso del Testing

4. Cierre de Pruebas

- Se **recolecta** la información de las actividades de prueba completadas para consolidar.
- Se **verifican los entregables** y que los defectos hayan sido corregidos.
- Se **evalúa** como resultaron las actividades de testing y se analizan las lecciones aprendidas.



Rol del tester...

- ▣ Ser integrante del equipo de proyecto e interactuar en los casos que lo requiera con el equipo de desarrollo (tanto internamente como con el cliente)
- ▣ Conocer los requerimientos (funcionales y no funcionales), workflows y caminos alternativos
- ▣ Conocer los procesos y las mejores prácticas del negocio
- ▣ Armar y generar los juegos de datos de prueba positivos y negativos.
- ▣ Considerar los "escenarios" en el armado de las pruebas.
- ▣ Armado y gestión de ambientes de prueba.
- ▣ Analizar y mantener la **trazabilidad** bidireccional entre el requerimiento, casos de uso, caso de prueba y los resultados obtenidos
- ▣ Ejecutar las actividades de tester, emulando los distintos tipos de roles de usuarios siguiendo las estrategias de testeo diseñadas.
- ▣ Capacitar y dar soporte interno en los casos que se requiera

Rol del desarrollador

- Implementa requisitos
- Desarrolla estructuras
- Diseña y programa el software
- Su éxito consiste en la creación de un producto

Rol del tester...

- Diseñar, desarrollar, ejecutar, retroalimentar y registrar los resultados del Test Plan.
- Crear los test cases basados en requerimientos del negocio así como en sus casos de uso, en base a una varias técnicas o estrategias de testing.
- Encontrar, reportar y realizar seguimiento de errores.
- Administrar el ciclo de vida de los errores
- Generar de información/documentación que respalde las pruebas.
- Analizar y reportar los resultados de las Pruebas.
- Conocer los requerimientos y su trazabilidad para asegurar los niveles de satisfacción acordados.

□ Percepción

- «La actividad del desarrollador es **Constructiva**»
- «La actividad del tester es **Destructiva**»

□ Error!!!!!!

- Las Pruebas también constituyen una actividad «**constructiva**», su propósito es la eliminación de defectos de un producto!!

Características de un buen tester

- ❑ **Curioso, Perceptivo, atento a los detalles (no todo error se manifiesta de forma evidente)**
 - ❑ Con el objeto de comprender los escenarios prácticos del cliente
 - ❑ Con el objeto de Poder analizar la estructura de la prueba
 - ❑ Con el objeto de descubrir detalles de dónde se puedan manifestar fallos

- ❑ **Escéptico y con actitud crítica**
 - ❑ Los objetos de prueba contienen defectos. Usted solo debe encontrarlos
 - ❑ No creer todo lo dicho por los desarrolladores
 - ❑ No se debe temer al hecho de se pudieran detectar defectos de importancia que pudieran tener un impacto sobre la evolución del proyecto

Características de un buen tester

▣ **Aptitudes para la comunicación**

- ▣ Necesarios para llevar «malas noticias» a los desarrolladores
- ▣ Necesarias para vencer estados de frustración
- ▣ Tanto cuestiones técnicas como prácticas, relativas al uso del sistema, deben ser entendidas y comunicadas
- ▣ Una comunicación positiva puede ayudar a evitar o facilitar situaciones difíciles
- ▣ Para establecer una relación de trabajo con los desarrolladores a corto plazo.

▣ **Experiencia**

- ▣ Factores personales influyen en la ocurrencia de errores
- ▣ La experiencia ayuda a identificar donde se pueden acumular errores.

TESTLINK



- Testlink es una herramienta para la administración de casos de prueba
- La herramienta nos ofrece:
 - Soporte para diferentes productos
 - Creación de diferentes roles con distintos permisos para los integrantes del equipo de testing
 - Creación ilimitada de carpetas en forma de arbol (llamadas testsuites) para una mejor organización y agrupamiento de los casos de prueba
 - Versionado de casos de prueba
 - Ejecución de los casos de prueba por versión del software bajo prueba
 - Creación de testplans para la ejecución
 - Asignación de casos (para ejecución) a los usuario. De modo que en un mismo test plan podemos tener varios testers trabajando y ellos sólo ven sus casos de prueba asignados
 - Soporte para linkear casos con requerimientos/especificaciones
 - Generación de distintos tipos de reportes: generales del producto, por testplan, gráficos, queries específicas, etc...
 - Posibilidad de mandar los reportes por mail al tester

Testlink – Estructura general

- El sitio, internamente, posee tres pilares fundamentales:
 - Proyecto de prueba (Test Project),
 - Plan de Pruebas (Test Plan) y el Usuario (User).
 - Todos los demás datos son relaciones o atributos de ellos. En primer lugar, se definirán los términos que se utilizaran en la documentación y en el mundo de las pruebas. Luego los pasos necesarios para registrarse como usuario y el logueo.

Testlink – Terminología básica

- ▣ **Caso de Prueba (Test Case):** Describe una prueba a través de los resultados esperados y los pasos (acciones, escenarios). Los Test Case son la pieza fundamental de TestLink.
- ▣ **Suite de Casos de Prueba (Test Suite):** Organiza los casos de prueba en unidades lógicas. En versiones anteriores, eran llamados Componentes y Categorías.
- ▣ **Plan de Pruebas (Test Plan):** Se crea cuando se desea ejecutar casos de prueba. Puede estar compuesto por uno o varios Test Projects. El Test Plan incluye Builds (Construcciones), Milestones (Hitos), asignación de usuarios y resultados de las pruebas.
- ▣ **Usuarios (Users):** Cada usuario tiene un papel que define las características disponibles de TestLink que puede utilizar.
- ▣ **Proyecto de Prueba (Test Project):** Es un componente que siempre existirá en TestLink y puede ser sometido a muchas versiones diferentes. Un Test Project incluye Pruebas de Especificación con casos de prueba, requerimientos y palabras claves. Todos los usuarios, dentro del proyecto, tienen un perfil definido.

Errores - Bugs

- No todos los defectos del sistema son causados por errores de código.
- Una fuente común de defectos es causada por la **falta de requerimientos** o requerimientos pobres.
 - Esto sucede por ejemplo con los problemas de requerimientos en aspectos no funcionales, escalabilidad, mantenibilidad, usabilidad, etc.
- Las **fallas** ocurren en el siguiente proceso: un programador comete un error, que resulta en un defecto en el código. Fuente. Si este defecto es ejecutado, en algunas ocasiones el sistema produce resultados incorrectos causando la falla.
- Un simple defecto puede resultar en un amplio rango de síntomas.
- Un **bug** es un error o un defecto en el software que hace que un programa funcione incorrectamente. A menudo los bugs son causados por conflictos del software cuando las aplicaciones intentan funcionar.

Como reportar un bug

- Existen programas llamados **bug trackers** que sirven para reportar bugs o defectos en un sistema.
- Alguno de estos bugtrackers son **Jira**, **redmine**, **mantis**, que son gestores de proyectos y poseen un sistema de tickets para reportar fallas y de esta forma los desarrolladores puedan verlo y corregirlo.
- El reporte debe cumplir con una serie de pautas que son las siguientes:
 - Asociación al proyecto
 - ID único
 - Título
 - Descripción amable explicando el bug
 - Pasos para reproducir el bug
 - Resultado actual
 - resultado esperado
 - Entorno de prueba (SO, browsers)
 - Screenshot / Video

Ejemplo de Bug

Search - Overflow in the search results

Añadido por Dinno Admin hace menos de 1 minuto.

Estado: Nueva
Prioridad: Normal
Asignado a: -
Categoría: -
Versión prevista: -

Fecha de inicio: 2013-01-07
Fecha fin:
% Realizado:  0%
Tiempo dedicado: -

Descripción

 Citar

The search results have overflow when a user insert a long text string

Steps:

- 1.- Go to: <http://qatraining.comli.com>
- 2.- In the search box insert:


"shfrsjkdhfskjdfhksjdhfskjdhfsjkdhfksjdfjhkhkjkhkjkhkjkhkjdhfsjkdhfksjdfhksdjfhsk"

- 3.- Click on "Buscar" button.

Actual Result: The system displays the text string overflow

Expected Result: The system should be display the text string correctly

Please, see the attached screenshot

 [bug.png](#) (64,411 KB)  Dinno Admin, 2013-01-07 12:07

Ejemplo de Bugs en un sitio Web

□ HTML

- 1.- Etiquetas mal cerradas
- 2.- Compatibilidad de HTML 5 entre browsers

□ Javascript

- 1.- Variables no definidas
- 2.- Problemas de Sintaxis
- 3.- problemas en sitios lentos
- 4.- problemas cuando no está habilitado

□ Seguridad

- 1.- XSS
- 2.- SQLi - Blind SQLi
- 3.- Problemas de validación
- 4.- LFI
- 5.- RFI

Ejemplo de Bugs en un sitio Web

□ **Flash**

- 1.- Problemas cuando no está activado
- 2.- Problemas de carga
- 3.- Problemas de llamadas recurrentes
- 4.- Problema en navegabilidad en móvil

□ **Errores HTTP**

- 1.- 404: links rotos
- 2.- 301/302: Redirecciones incorrectas
- 3.- 403: Servicios que no funcionan por faltas de permisos
- 4.- 500: Servicios caídos

□ **Cookies**

- 1.- Cookies sin vencimientos
- 2.- Cookies que no están encriptados
- 3.- problemas de cookies de diferentes dominios (?)

Ejemplo de Bugs en un sitio Web

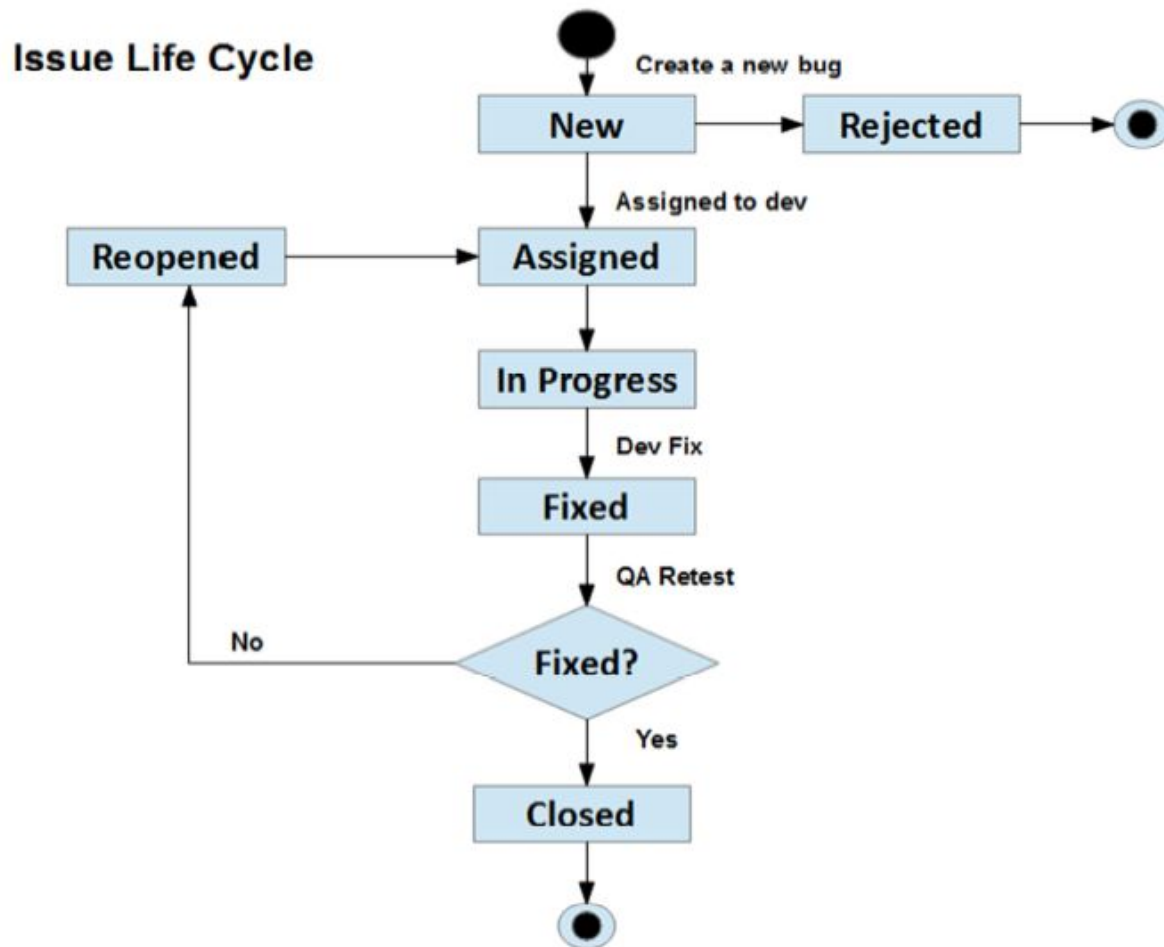
❑ Rendimiento del sitio

- ❑ 1.- Archivos pesados como imágenes >> Usar sprites
- ❑ 2.- muchos archivos js y css >> js/css minifying
- ❑ 3.- Peticiones constantes al servidor

❑ Estilos

- ❑ 1.- Colores y fuentes incorrectas
- ❑ 2.- problemas de layout
- ❑ 3.- problema de tamaño de los objetos

Ciclo de vida de un Bug



MÉTODOS DE PRUEBA

▣ Pruebas estáticas y dinámicas

· Pruebas estáticas

- ▣ Las pruebas estáticas se basan en el examen **manual**, también llamado revisiones, y en el análisis automatizado, o análisis estático del código o de cualquier documentación del proyecto sin ejecutar el código.
- ▣ Las **revisiones** constituyen una forma de probar los productos de trabajo del software, incluyendo código, y pueden realizarse antes de ejecutar las pruebas dinámicas.
- ▣ Los **defectos** detectados durante las revisiones al principio del ciclo de vida (ej.: en requerimientos) a menudo son más baratos de eliminar (ej.: omisiones en los requerimientos).
- ▣ Este tipo de revisiones pueden realizarse tanto manualmente como mediante herramientas de soporte.
- ▣ La principal actividad manual consiste en **examinar** un producto de trabajo y hacer comentarios al respecto.
- ▣ Cualquier **producto** puede ser objeto de una revisión, como por ejemplo, requerimientos, código, test cases, scripts.

• Pruebas estáticas

- **Ejemplo** de este tipo de pruebas estáticas son las **revisiones** por pares o peer reviews . Se pueden realizar revisiones por pares a bugs reportados, a test cases, etc.
- Los **beneficios** de realizar estas revisiones incluyen como ya mencionamos,
 - *“la detección y corrección temprana”* de defectos,
 - *el desarrollo de mejoras en el proceso y en la productividad,*
 - *el ahorro de tiempo y dinero, etc.*
- Entre los **defectos** típicos que resultan más fáciles localizar en las pruebas estáticas se incluyen:
 - desviaciones de estándares.
 - Defectos de **requerimientos**.
 - Defectos de **diseño, mantenibilidad** insuficiente, y especificaciones de **interfaz** incorrectas

• Pruebas Dinámicas

- La ejecución de pruebas dinámicas o análisis dinámico se refiere al conjunto de pruebas que requieren la **ejecución del código** para determinar cómo es su funcionamiento a lo largo del tiempo.
- En la prueba dinámica el software en realidad debe ser **compilado y ejecutado**; consiste en trabajar con el software, dando valores de entrada y comprobar si el resultado es el esperado. Estas son las actividades de validación.
- Pruebas **unitarias, pruebas de integración, pruebas de sistema y pruebas de aceptación** son algunas de las metodologías de pruebas dinámicas.
- Se parte de test cases específicos por la ejecución del objeto de prueba o programas en ejecución.
- Existen variedad de estrategias de testing que se describen más adelante y que son parte de este conjunto de pruebas dinámicas

