U-tad
CENTRO UNIVERSITARIO
DE TECNOLOGÍA
Y ARTE DIGITAL

Verificación del Software

U-tad
CENTRO UNIVERSITARIO
DE TECNOLOGÍA Y ARTE DIGITAL

U-tad CENTRO UNIVERSITARIO DE TECNOLOGÍA Y ARTE DIGITAL

Verificación del Software

1. Fundamentos de Calidad del Software

Verificación del Software

Tema 1. Fundamentos de Calidad del Sofware

Tema 2. QA en el SDLC

Tema 3. Revisión y pruebas

Tema 4. Técnicas y herramientas

Tema 5. Gestión de las pruebas

Fundamentos de Calidad del Software

1.1. ¿Qué es Calidad?
1.2. Calidad en sw
1.3. Conceptos básicos: QA. QC, testing
1.4. Verificación del software o testing
1.5. Los siete principios del testing
1.6. El testing como proceso

1.1 ¿Qué es Calidad? Calidad en el software

¿Qué es Calidad?

Antes de empezar …

Debate: ¿Qué es calidad?

Introducción

El diccionario recoge 10 acepciones para la palabra “calidad”. Va desde una característica que permite acreditar el *valor* de algo o alguien, un estado y un sinónimo de superioridad y excelencia.
También una, un poco forzada, que habla de “Adecuación de un producto o servicio a las características especificadas” es lo que formalmente persigue el control de calidad.
Como profesionales tenemos que lidiar con esa **percepción** (valor superior de algo) y esa **definición** (ajustarse a lo especificado). Como veremos a lo largo de esta asignatura, aunque las técnicas nos ayudan con la segunda, no podemos descuidar la primera, porque es lo que la sociedad, usuarios, clientes, … tienen en mente cuando usan la palabra “calidad”.
Aunque podamos pensar que como profesionales, basta con ajustarse a criterios estrictos y definidos, no olvidemos nunca que al final, tratamos con personas.

¿Qué es Calidad?

La dualidad de la calidad

De forma intuitiva acaban surgiendo dos grandes dimensiones de la calidad:

- **QUÉ**, o el producto "correcto"
- **CÓMO**, producto construido "correctamente"

Normalmente entendemos un producto de calidad como "libre de errores" pero hay más cosas:

- Experiencia ofrecida (tiempo de respuesta, usabilidad, atractivo)
- Cumplimiento de normativa
- Consumo de recursos, viabilidad económica
- "Estar a tiempo"

Teniendo en cuenta que esas otras dimensiones evolucionan en el tiempo (lo que hace años era excepcional, hoy es el mínimo exigible, etc).

¿Qué es Calidad?

Persistencia de la calidad

La calidad es una característica curiosa. Para empezar no es fácil de modular: si invertimos más dinero podemos entregar más cosas, si invertimos menos, el producto entregará menos funcionalidad. Esto no funciona con la calidad: nadie está dispuesto a ahorrar en calidad, y la mala calidad permanece, si no se corrige.
A un cliente puede no gustarle que el producto tenga menos funcionalidad de la esperar o se entregue más tarde de lo previsto, pero tiende a vivir con ello. Nadie se conforma con un producto de baja calidad.

"*The bitterness of **poor quality** remains long after the sweetness of low price is forgotten.*“

Benjamin Franklin

“Ahorrar en calidad” es una pésima estrategia. La calidad debe ser central, indiscutible y estratégica.

¿Qué es Calidad?

Relación entre Calidad y esfuerzo

https://youtu.be/x9wn633vl_c

¿Qué es Calidad?

Definición

"*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*", Roger Pressman.

- "*Proceso eficaz de software*": foco en cómo hacemos las cosas, y los procesos, herramientas, prácticas, técnicas.
- "*Un producto útil*", se refiere a cumplir las expectativas (requisitos) de los usuarios finales con algo confiable y libre de errores.
- "*Proporciona valor medible a quienes lo producen y a quienes lo utilizan*", proporciona beneficio para quien lo hace (actividad económica) y para quien lo recibe, es decir, no sólo el producto es correcto y está hecho de la forma adecuada, es que aporta valor a quien lo recibe.

Atributos de un producto de Calidad

Dentro de los estándares "SQuaRE (*Software product Quality Requirements and Evaluation*)", el ISO/IEC 25010 define 8 características para un producto de calidad:

- Adecuación funcional: atributos que satisfacen necesidades explícitas e implícitas.
- Eficiencia: equilibrio entre los recursos consumidos y el rendimiento ofrecido.
- Compatibilidad: coexistencia e interoperatibilidad con otros componentes de su entorno.
- Usabilidad: lo que incluye facilidad de uso, aprendizaje o accesibilidad.
- Fiabilidad: que incluye la tolerancia a fallos, disponibilidad o capacidad de recuperación.
- Seguridad: como integridad, autenticidad o no repudio.
- Mantenibilidad: capacidad de ser modificado, y probado, así como reusabilidad, o modularidad.
- Portabilidad: adaptabilidad, facilidad de instalación o de ser reemplazado.

Todo ello define una calidad con múltiples facetas, que va más allá de cumplir requisitos, que es como muchas veces se aplica el concepto en el mundo del desarrollo software.

Dilema de la Calidad

Se refiere a si un producto debe tener una calidad absoluta o a ser “suficientemente bueno” *(good enough).*

Aunque pensemos que es impensable entregar un producto con taras o carencias, ocurre continuamente, especialmente en el mundo del software. Es raro el producto servicio que no contiene “*known issues*” o problemas y carencias detectados no solucionados en el momento de entregar a los clientes.

Normalmente se trata de problemas menores que no afecten al uso del producto o a la reputación de la organización. Pero a veces son verdaderas carencias que se pueden ver “perdonadas” por la presencia de otras características que aporten novedad o utilidad (por ejemplo, en el caso del primer iPhone).

Esto ocurre debido que hay factores en el lanzamiento de un producto que pueden hacer superar los condicionantes de calidad:

- Inversión necesaria para corregir todos los problemas detectados.
- Carácter novedoso o diferenciador que puede tapar las carencias.
- Tiempo para salir al mercado, bien para cumplir una fecha límite, para llegar antes que la competencia o para poner el producto en manos de sus clientes y recuperar inversión.

¿Tiene coste la Calidad?

Y si lo tiene ¿cuál o cuáles?

El coste de la Calidad (1)

La calidad tiene coste

La calidad tiene un coste, de la misma forma que la ausencia de calidad también lo tiene.
En su extremo, el coste de la ausencia de calidad es, al menos, similar al de la inversión realizada para desarrollar (y fabricar si es físico) el producto o servicio. Puede ser aún mayor si afecta a la reputación de la empresa impactando en sus ingresos futuros hasta hacerla inviable. Como es difícil de cuantificar (anticipadamente) hay quien lo ignora, lo que es una mala solución y además transmite la idea de que la Calidad es sólo un coste y no una inversión que reduce el riesgo:

Coste de Calidad < Coste de falta de calidad

Obviamente, si la relación se invierte, no tiene sentido preocuparse por la calidad del producto (cosas que ocurren, por ejemplo, en caso de escasez o necesidad extremas).
Esto también se puede expresar como :

Costes internos < Costes externos

Donde costes internos son los dedicados a prevenir errores en el producto o servicio y costes externos los relacionados con encontrar los errores **después** de entregado a sus clientes y usuarios.

El coste de la Calidad (2)

Coste de asegurar la calidad

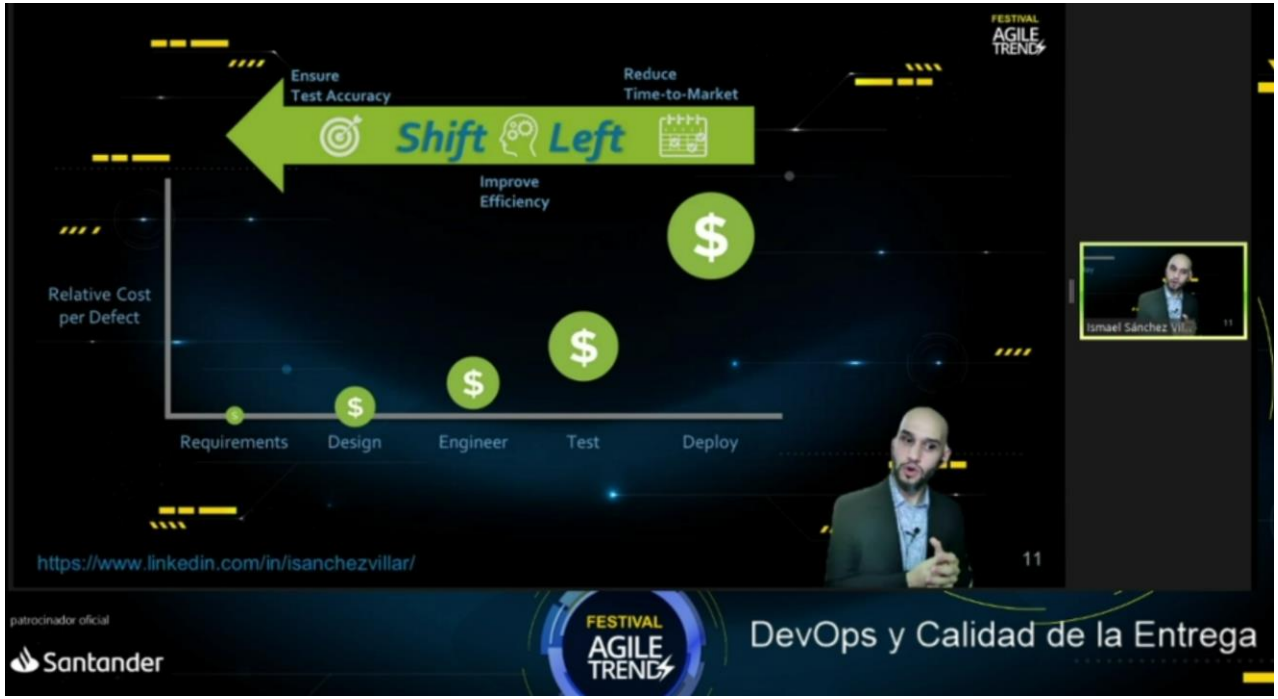
Se divide en tres grandes apartados:

- Costes de prevención
  - Coordinación
  - Desarrollo de modelos de requisitos y diseño
  - Planificar pruebas
  - Capacitación para hacer las pruebas
- Costes de evaluación
  - Revisiones técnicas
  - Medir la evaluación
  - Probar y evaluar
- Costes de fallo (internos, antes de entregar el producto)
  - De reparar fallos
  - De errores colaterales
  - De medir proceso y resultados

Donde se piensa normalmente que se aplica la calidad

El coste de la Calidad (3)

Coste de la calidad crece con el tiempo



"El testing no aporta valor" Ismael Sánchez en Agile Trends 2021

Verificación del Software

1.2 Calidad en el software

Calidad en el software (1)

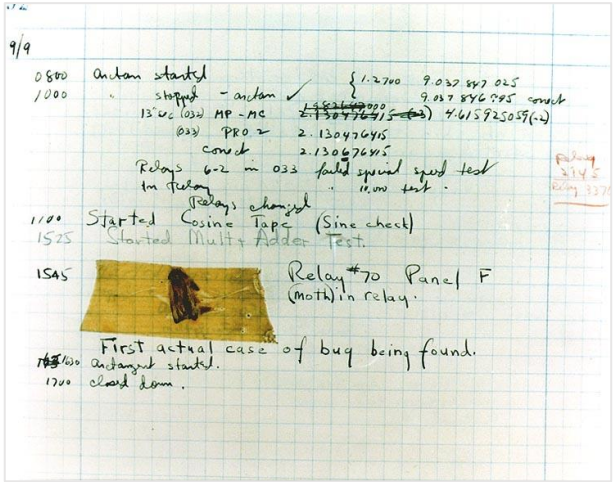
Bugs

En los primeros años de la informática (40 a 60 del siglo pasado) la componente fundamental era el hardware, por eso la calidad aplicada era la propia de los sistemas de fabricación industrial.

El termino “bug” nace en 1947 cuando Grace Hopper (pionera del mundo de computación y la programación) encontró un auténtico “bicho” que estaba provocando un fallo en un ordenador Mark II.

Desde los años 50, el término describe cualquier error en el sw o hw, y “debugging” el proceso de eliminarlo.

La primitiva calidad de sw consistía en un proceso de prueba y error llevado a cabo por programadores (en realidad programadoras, que fueron las pioneras en los primeros años).



Calidad en el software (2)

Ingeniería

A medida que el sw ganaba más relevancia y se convertía en el elemento clave de cada vez más componentes críticos, sus problemas y carencias eran más evidentes, hasta el punto que en 1968, la OTAN (el mayor contratista de sw del mundo de la época) convocó una conferencia con los principales expertos mundiales (europeos en su mayoría entonces). Los problemas del sw hace 50 años era:

- Proyectos que superaban las fechas previstas.
- Coste superior al estimado.
- Desalineamiento entre las expectativas y el resultado.
- **Baja calidad del sw (bugs)**.

La solución es la disciplina que conocemos como "Ingeniería del sw" en la que calidad se consideraba un componente fundamental. Como se inspiraba en las técnicas de fabricación industrial (clásicas, previas al TPS y otras innovaciones), la calidad se convertía en una etapa del proceso, en la que se revisaba el producto ya hecho para buscar "bugs".

El modelo waterfall o predictivo sigue vigente en muchas organizaciones que continúan haciendo pruebas al final, cuando el producto está ya hecho (y es más costoso arreglarlo).

Calidad en el software (3)

Calidad e2e y automatización

Desde el principio algo no acababa de encajar en el nuevo modelo “waterfall”: esperarse al producto construido para hacer las pruebas suponía que la corrección era mucho más cara, si se hacía ese control y sus correcciones mucho antes.

Desde los años 80, una serie de modelos alternativos se han sucediendo para proponer otras formas mejores de resolver los problemas del software: Espiral, RAD, DSDM, …

Los marcos de trabajo ágiles, inspirados en Lean y los métodos de trabajo de la industria japonesa (TPS) tratan la calidad como un elemento integral que no se relega a una fase determinada.

Las grandes innovaciones desde el punto de vista de la calidad vienen de la mano de XP (2000) y DevOps. De forma muy simplificada son:

- Test-Driven Development: primero se define la prueba y luego se construye el código que la cumple. Los tests son parte del trabajo de programación.
- Automatización: incluyendo las pruebas. En lugar de ser un hecho puntual y manual, las pruebas son continuas y automatizadas incrementando la confianza.
- Responsabilidad e2e: el equipo que construye el sw no se despreocupa de él una vez ha terminado, si no que se responsabiliza de su correcto funcionamiento siempre.

1.3 Conceptos básicos de la Calidad del software

Conceptos básicos de la calidad en el sw

Hemos visto hasta ahora una definición general de qué es calidad y conceptos relacionados como el coste que tiene.
También hemos visto como se ha aplicado en el mundo de la fabricación industrial y como saltó de ahí al mundo del software.
En este apartado entraremos en conceptos básicos dirigidos específicamente a la calidad en el software como son QA yQC, testing, fallo, error y otros muchos.
Dedicaremos un espacio a hablar de la mentalidad adecuada para gestionar adecuadamente la calidad en el software.

Quality Assurance y Quality Control (1)

Son dos conceptos muy importantes que hay entender y diferenciar:

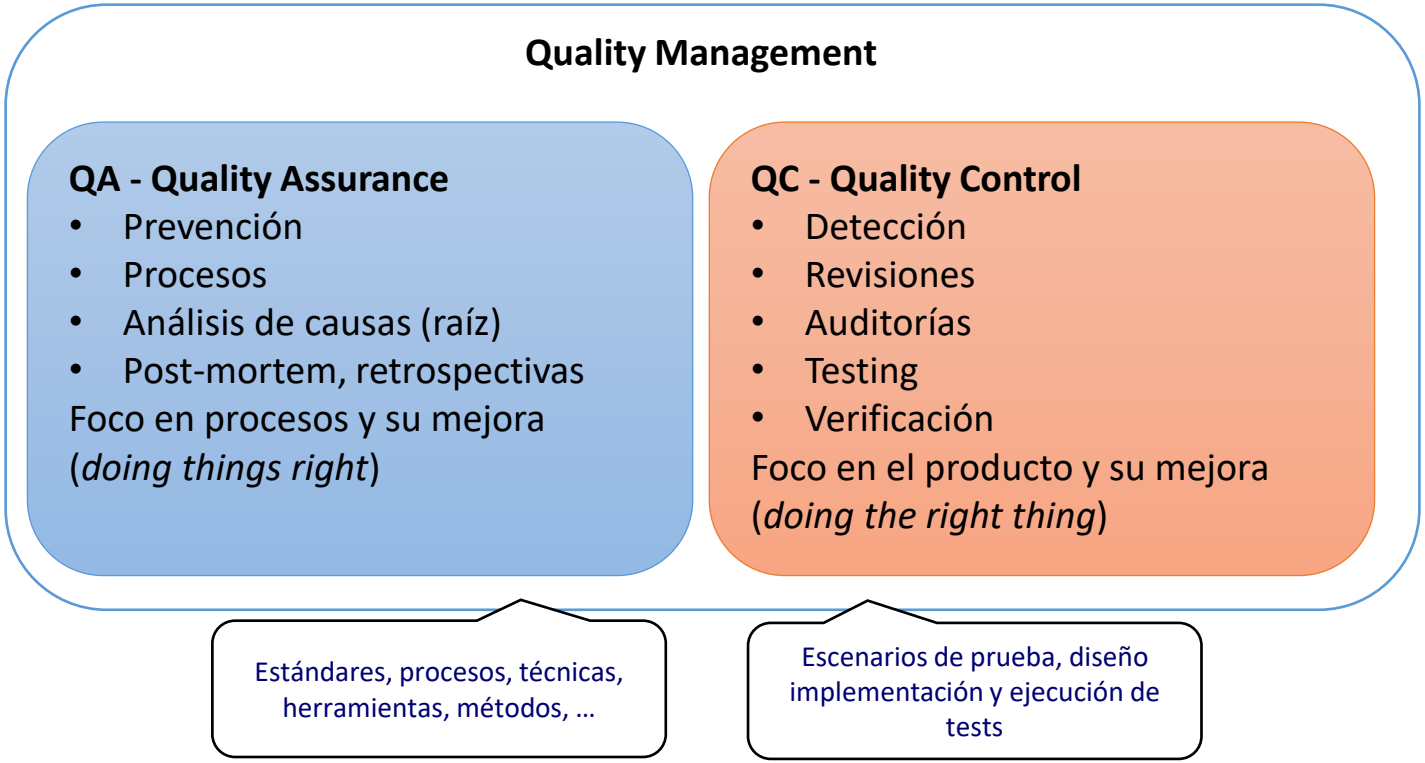
- QA, Quality Assurance o Aseguramiento de la Calidad:
  - Es una serie de técnicas y procesos centrados en la **prevención** de defectos en el software.
- QC, Quality Control o Control de la Calidad:
  - Técnicas que ayudan a verificar y validar la calidad en el software.

Algunos autores los consideran dos áreas dentro de la gestión más general de la calidad Mientras que otros hacen que la QC sea parte de la QA. En cualquier caso hay que tener muy claro que:

QA ≠ QC

Cuidado: a veces se usan las siglas "QA" como sinónimo de "Calidad" especialmente cuando se habla de equipos, departamentos o perfiles. En esos casos no se refieren al concepto de Quality Assurance, si no al de testing o verificación del software.
También hay autores que usan un único concepto para englobarlo todo sin diferenciación (SQA, *Software Quality Assurance*)

Quality Assurance y Quality Control (2)



Nomenclatura del testing

Errores, defectos y fallos

Error:

Acción humana que produce un resultado incorrecto. Un error provoca defectos.

Defecto (a veces también *fault* o *bug*):

Una deficiencia o imperfección del producto que no se ajusta a los requisitos o especificaciones. Un defecto puede provocar un fallo (pero no necesariamente).

Fallo

Funcionamiento incorrecto del sistema o uno de sus componentes. No tiene porqué deberse necesariamente a un defecto, puede ser motivado por otras causas.

Algunos autores llaman error a lo detectado antes entregar y defecto a lo detectado después.

¿Error, defecto o fallo?

1. Si uso emojis en la password, la aplicación se cae.
2. La query de la base de datos está mal hecha pero los resultados que devuelve son válidos.
3. El software cumple el requisito de 22 números en el IBAN pero en realidad son 24.
4. El ordenador de vuelo da de vez en cuando un “divide by zero” y hay que apagarlo y encenderlo. Nunca ha habido un accidente.
5. Mi calle no sale en la lista de direcciones para el envío del pedido.
6. En el juego, te puedes caer por debajo del suelo.
7. El cálculo de cambio de siglo está mal programado en la app Android.
8. Cuando el nivel de gasolina es muy bajo, deja de calcular la autonomía.

¿Qué provoca los errores?

Casi siempre son humanos

Más probables:

Deficiencias en la forma de expresar requisitos, complementadas con hacer asunciones sin contrastar su validez.

Problemas de comunicación dentro del equipo y con los stakeholders: falta de claridad, diferencias de interpretación, ambigüedad, …

Falta de foco y atención, interrupciones, confusiones, …

Falta de conocimientos o experiencia en las técnicas y herramientas.

Exceso de presión por fechas de entrega agresivas.

Complejidad del problema o la solución

Errores humanos al escribir o marcar opciones.

Otros tipos de errores:

Magnetismo, contaminación, radiación, … pueden dar lugar a cambios **físicos** en los soportes del software que podrían acabar provocando errores ocasionalmente. En general, son poco probables.

Detectar fallos no es detectar errores

Los tests deben tener su propia QA y QC

Falsos positivos:

Test que falla en vez de pasar.
Esto da lugar a esfuerzo innecesario y pérdida de confianza.

Falsos negativos:

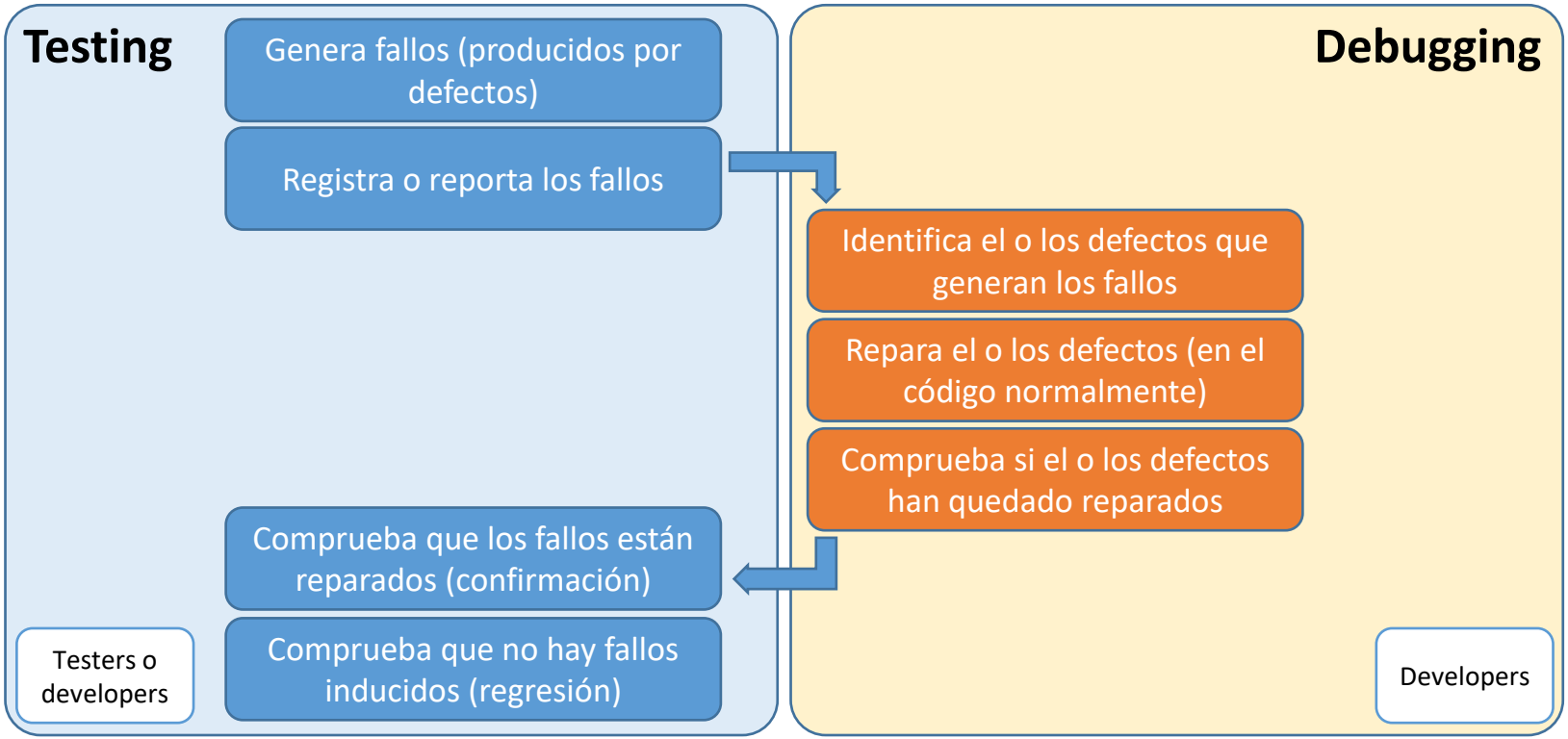
Test que pasa en vez de fallar.
Esto supone que se da por bueno software con defectos.

Las razones para esto pueden ser muchas:

- Mal diseño del test, por ejemplo interpretando erróneamente los requisitos.
- Mala implementación, seleccionando herramientas o conjuntos de datos inadecuados.
- Mala ejecución o defectos en las condiciones de la prueba, en los datos usados, en la interpretación de los resultados, o los pasos de la prueba.

Testing y debugging

Debugging es el proceso de encontrar y eliminar las causas de los fallos en el software.



Test case

Un test case o caso de prueba es la unidad básica del testing

Contiene:

- **Precondiciones**:
  - Aspectos o acciones necesarios antes de iniciar el test
- **Entradas**
  - Datos, condiciones, acciones que definen la prueba
- **Resultados esperados**
  - Los datos, acciones, comportamientos que determinan que el objeto de la prueba ha superado (pasado) el test
- **Postcondiciones**
  - Acciones tras las prueba, normalmente necesarias para restablecer el entorno y/o el objeto de prueba a un estado que permita nuevas pruebas

Trazabilidad

Asegura la integridad y completitud de las pruebas

En el mundo del testing hay que asegurar muy bien la trazabilidad:

- Qué pruebas validan qué requisito
- Qué requisitos se validan con una prueba
- Y las versiones de cada uno de ellos
- Y los resultados obtenidos en un momento determinado al ejecutar un test de una versión sobre un objeto de otra

Por ejemplo si diseñamos un test para las funcionalidades ofrecidas por un determinado objeto software, no sirve de nada según probando con él si ha cambiado la definición e implementación del objeto software.
Los tests no son estáticos, evolucionan con el software, y a veces evolucionan aunque el software no cambie. Por eso tenemos que ser muy rigurosos con la trazabilidad entre tests casos, objetos, requisitos, condiciones de prueba, … y en general todos los elementos involucrados.
Un ejemplo clásico de “catástrofe” en las pruebas es invertir esfuerzo es ejecutar un paquete de pruebas que no se corresponde con el software que se está probando.

1.4 Verificación del software o testing

Verificación del software (testing)

En esta asignatura, vamos a centrarnos en las actividades de Quality Control, que abarcan tanto la verificación como la validación, que son dos aspectos distintos:

- **Verificación**:
  - Se trata de determinar si el software cumple con las especificaciones que lo definen. Esto implica analizar artefactos como diseños, código, … para determinar que se ajustan a lo especificado. Tiene mucho que ver con QA y los procesos.
- **Validación**:
  - Se trata de determinar que el comportamiento del software satisface las necesidades de los stakeholders. Esa validación se puede realizar a partir de los requisitos (interna) o haciendo que los propios stakeholders comprueben el software (tests de aceptación).

La Verificación y Validación o V&V se lleva a cabo por medio del **testing**, que es el conjunto de técnicas que trabajaremos en esta asignatura.

Verificación y Validación

"*Una causa común de desastre en el desarrollo de software es que el producto final sea precisamente lo que el cliente pidió originalmente*".

The Economist

The Economist

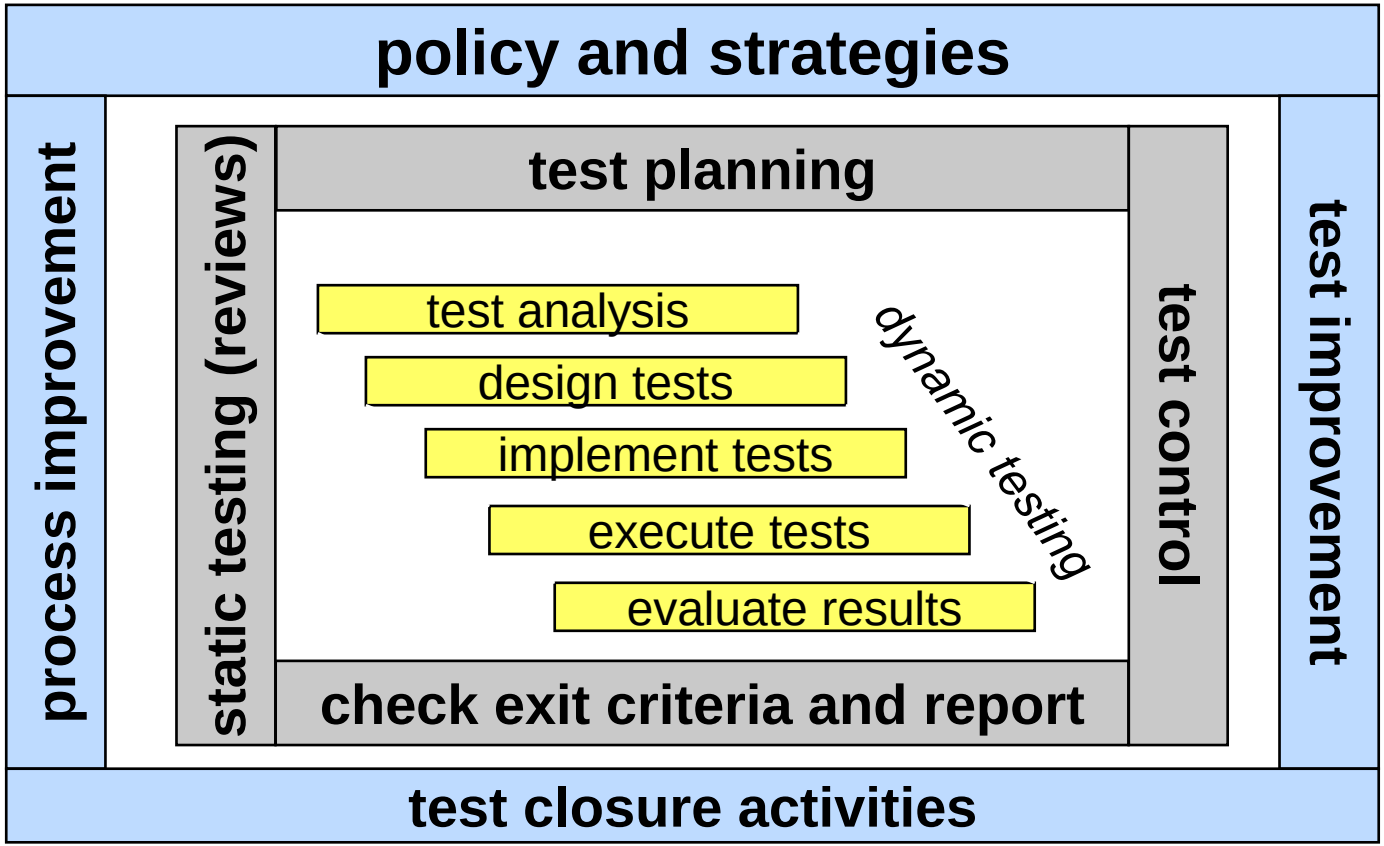
"La agilidad cuenta"

http://www.economist.com/node/779429

¿Quiénes son los stakeholders?

¿Y los shareholders?

Testing (1)



Modelo de Testing según ISTQB

Testing (2)

Reducción de riesgo e incertidumbre

Dado que el software es ya prácticamente ubicuo, las consecuencias de su mala calidad afectan a:

- Infraestructuras
- Salud de las personas
- Seguridad
- Economía
- Gobierno
- Educación
- Información
- …

Con impactos cada vez más amplios y más graves. Es inevitable dedicar atención y esfuerzo a su calidad, lo que se traduce en actividades de testing.

El Testing es la forma de **evaluar la calidad** del software y de **reducir el riesgo** de fallo cuando se use.

El testing no da certezas, pero sí reduce incertidumbre.

Testing (3)

Objetivos del testing

- Prevenir defectos por medio de la evaluación (estática) de los productos del trabajo (diseños, requisitos, código)
- Verificar que se han cumplido los requisitos
- Validar que el producto se comporta tal y como esperan los stakeholders
- Generar confianza, así como reducir incertidumbre y riesgos sobre la calidad de los elementos probados.
- Proporcionar información para la toma de decisiones con respecto a la calidad del sistema ("*good enough*")
- Determinar la *compliance* (o cumplimiento) con estándares y requisitos legales y regulatorios
- Determinar la completitud de los elementos probados
- Encontrar defectos y fallos

Estos objetivos pueden depender del contexto, qué es lo que se está evaluando, el momento del ciclo de vida del sw, …

"Encontrar fallos" es sólo uno más de los objetivos y no necesariamente el más importante

Testing (4)

Dinámico y estático

Son las dos grandes modalidades y cada una tiene un propósito definido:

Testing dinámico:

- Requiere que se ejecute el software (se “ejecutan” los tests)
- Busca generar fallos
- La ejecución puede ser manual o automática
- El diseño es siempre manual
- Se suele basar en casos de prueba pero también puede ser libre (exploratorio)

Testing estático:

- No hay ejecución de software (ni de tests)
- Busca encontrar defectos
- Puede ser manual: revisión de requisitos, documentación, código
- O automática y se puede apoyar en herramientas de análisis (de código)

El testing colabora durante el SDLC

- El proceso de especificación de requisitos
  - Es el mejor momento para detectar defectos: es cuando tienen más impacto y su solución es más económica
  - Reduce el riesgo de que se construya funcionalidad incorrecta, innecesaria o difícil de evaluar.
- En el proceso de diseño
  - Mejorando el entendimiento compartido del diseño y de cómo verificarlo
  - Diseñando colaborativamente escenarios de prueba
- Durante la implementación
  - Mejorando la comprensión del código y de la forma de evaluarlo (tests)
  - Colaborando cada parte en el trabajo de la otra. Incluso mejor si no hay “partes”
- Verificando el producto antes de la entrega
  - Detectando fallos y ayudando a determinar su origen
  - Ayudando a eliminarlos
  - Realizando aportaciones constructivas a aspectos no funcionales

Incluso aunque haya un proceso altamente automatizado, es muy necesario el punto de vista del testing

La causa raíz

No nos conformamos

La causa raíz es la fuente última del defecto, de forma que si la eliminamos, el defecto desaparece (y el fallo cesa).

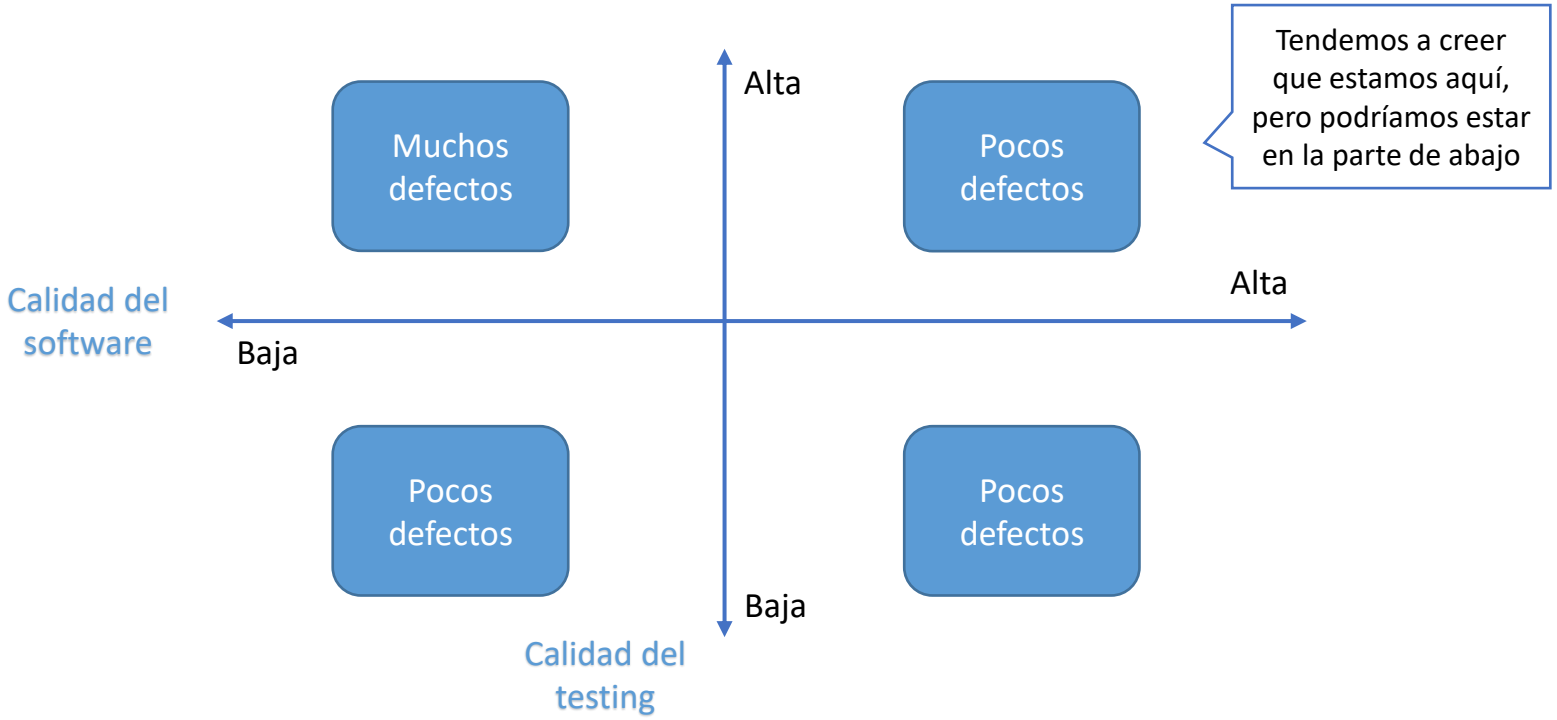
Encontrar la causa raíz es importante no sólo como forma de eliminar el defecto, si no que debe ayudar a mejorar el conjunto del proceso, ahorrando la aparición de defectos futuros. Es una oportunidad para la mejora continua. La prevención de defectos forma parte de la QA (Quality Assurance).

Por ejemplo, un defecto puede deberse a una mala interpretación de un requisito; que se interpretó mal porque no se contrastó; que no se pudo contrastar porque el Product Owner no estaba disponible para el equipo; que no lo estaba porque tenía que atender a varios proyectos. Todo esto debería llevar a reducir la carga de trabajo del PO y aumentar su foco y capacidad de atender al equipo.

Técnicas para descubrir la causa raíz: Isikawa, 5 porqués, o el Andon cord de Toyota

El testing necesita una visión crítica

Sin ella, podemos caer en un espejismo



1.5 Los siete principios del testing

De acuerdo con ISTQB, una de las principales organizaciones dedicadas a la formación y certificación en testing.

Principios del testing (1)

Las pruebas muestran la presencia de defectos, no su ausencia

PREGUNTA

Si todos los tests pasan correctamente ¿Queda el sw libre de defectos?

Si algunos tests fallan ¿Hemos encontrado todos los defectos?

Principios del testing (1)

Las pruebas muestran la presencia de defectos, no su ausencia

PREGUNTA

Si todos los tests pasan correctamente ¿Queda el sw libre de defectos?

Si hay tests que fallan ¿Hemos encontrado todos los defectos?

Por regla general, las pruebas sirven para **reducir la probabilidad** de que se entregue el software con defectos. Pero no la elimina.
No importa lo largo o completo que sea el proceso de pruebas: no podemos decir que no quedan defectos en el sw.
Al revés, el testing sólo puede probar que hay defectos con respecto a los que encuentra. No puede decir nada sobre los no detectados.

Principios del testing (2)

El testing exhaustivo es imposible

Hacer todas las pruebas posibles, con todas las variantes posibles de datos y condiciones es inviable por esfuerzo y tiempo. Sólo se puede conseguir puntualmente.

Por ello, el testing se centra en identificar y eliminar riesgos y probabilidad de fallo (casos y funciones críticos, cribado, priorización de escenarios y condiciones).

Principios del testing (2)

El testing exhaustivo es imposible

Hacer todas las pruebas posibles, con todas las variantes posibles de datos y condiciones es inviable por esfuerzo y tiempo. Sólo se puede conseguir puntualmente.

Por ello, el testing se centra en identificar y eliminar riesgos y probabilidad de fallo (casos y funciones críticos, cribado, priorización de escenarios y condiciones).

EJEMPLO:

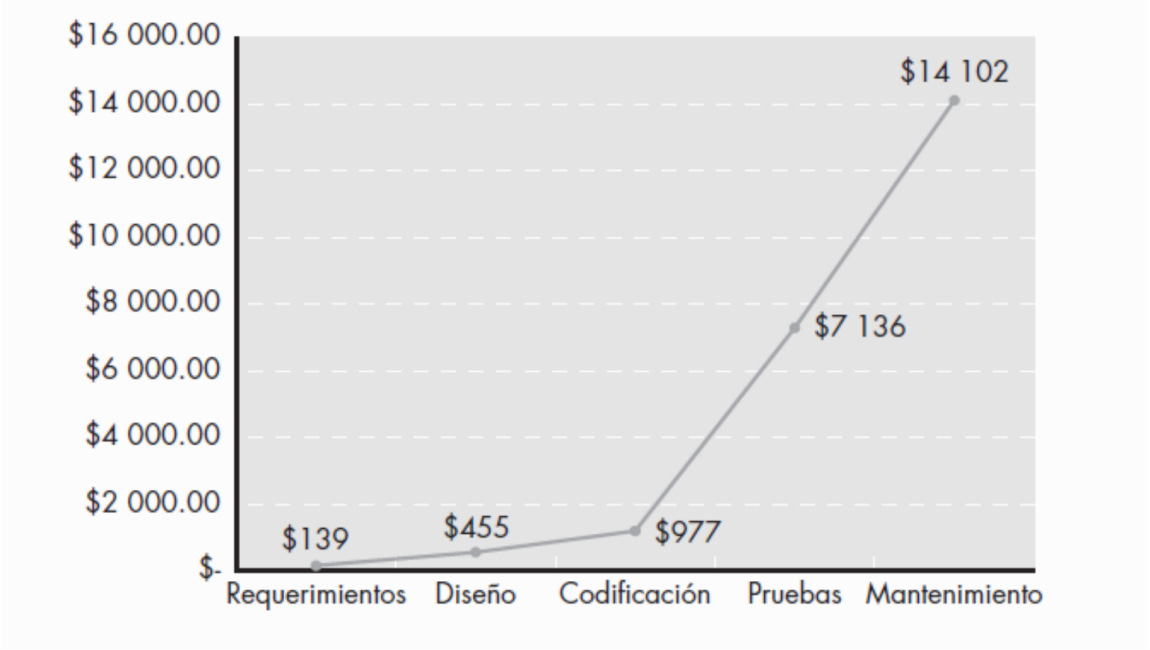
Tenemos que probar un sistema de programación agua caliente que entrega agua entre 10 y 90º, en incrementos de 0,5º; que se enciende en cualquier momento del día en saltos de 5'; y que puede estar funcionando hasta entre 5' y 24 horas en saltos de 5':

Casos posibles: 80*2*24*20*24*20 = 36.864.000
Si cada caso se prueba durante 5" : 8 años
Si cada caso se prueba durante 1" : 19 meses

Principios del testing (3)

Probar pronto ahorra tiempo y dinero

El coste de reparar el sw crece exponencialmente a medida que se avanza en su construcción.



Según Boehm, B. y V. Basili, "*Software Defect Reduction Top 10 List*", *IEEE Computer*, citado por Roger Pressman ("Ingeniería de software un enfoque práctivo")

Principios del testing (4)

Los defectos se agrupan

Los defectos no se distribuyen homogéneamente.
Normalmente, los defectos y problemas se concentran en partes concretas del sw.
Por esa razón, el foco (atención, esfuerzo, tiempo) se debe centrar en identificar esos puntos y en probarlos prioritariamente.
Ser capaces de predecirlos proporciona una gran ventaja.

Principios del testing (5)

Cuidado con la paradoja del pesticida

PREGUNTAS:

¿Deben encontrar defectos los tests?
¿Deben ser repetibles los tests?
Si volvemos a ejecutar un mismo test ¿aparecerán defectos nuevos?

Los tests están para detectar fallos. Un test exitoso es que el descubre un defecto, no el que "pasa".
La paradoja del pesticida es que puede eliminar ciertos insectos y malas hierbas, pero no a todos. Si usamos siempre el mismo, no terminaremos con las plagas (y además se adaptarán).
Esto implica que hay que revisar y actualizar tests, crear otros nuevos, ejecutarlos de distintas formas … porque el objetivo es detectar nuevos defectos, no conseguir que todos los tests funcionen.
Esta idea y su colisión la "mentalidad de desarrollador" es el origen de muchos de los problemas en proyectos y organizaciones.

Principios del testing (6)

El testing depende del contexto

No todas las pruebas son iguales porque no todos los productos son o se usan igual

PREGUNTAS:

¿Se te ocurren ejemplos de productos que requieran pruebas más completas y exhaustivas?
¿Y ejemplos de lo contrario?

El software crítico requiere procesos más formales y planificados.
El software comercial tiene que encontrar un equilibrio que no comprometa calidad ni rentabilidad.
Un producto nuevo en un mercado nuevo puede asumir más riesgos porque es más importante el *time to market* que la calidad estricta.

Principios del testing (7)

La falacia de la ausencia de error

PREGUNTAS:

¿Hay que arreglar todos los defectos que se encuentran?
¿Está bien entregar sw con defectos?
Si hemos arreglado todos los defectos que hemos encontrado ¿está el sw perfecto y libre de errores?

La detección y reparación de defectos por si sola no es suficiente, como no lo es cumplir estrictamente con los requisitos. Por ejemplo, nuestro producto puede ser funcionalmente correcto pero tener unos tiempos de respuesta o una experiencia de usuarios malos.
La labor de la calidad no termina en verificar que se han cumplido los requisitos explícitos.

Principios del testing: Resumen

1. **Las pruebas muestran la presencia de defectos, no su ausencia**
2. **El testing exhaustivo es imposible**
3. **Probar pronto ahorra tiempo y dinero**
4. **Los defectos se agrupan**
5. **Cuidado con la paradoja del pesticida**
6. **El testing depende del contexto**
7. **La falacia de la ausencia de error**

1.6 El testing como proceso

El proceso del testing

No hay una forma única de llevar a cabo el testing (de la misma forma que no la hay para el desarrollo). Pero sí hay una serie de actividades comunes. El proceso del testing consistirá en seleccionar:

- Qué actividades realizar,
- Cómo hacerlas
- Cuándo hacerlas

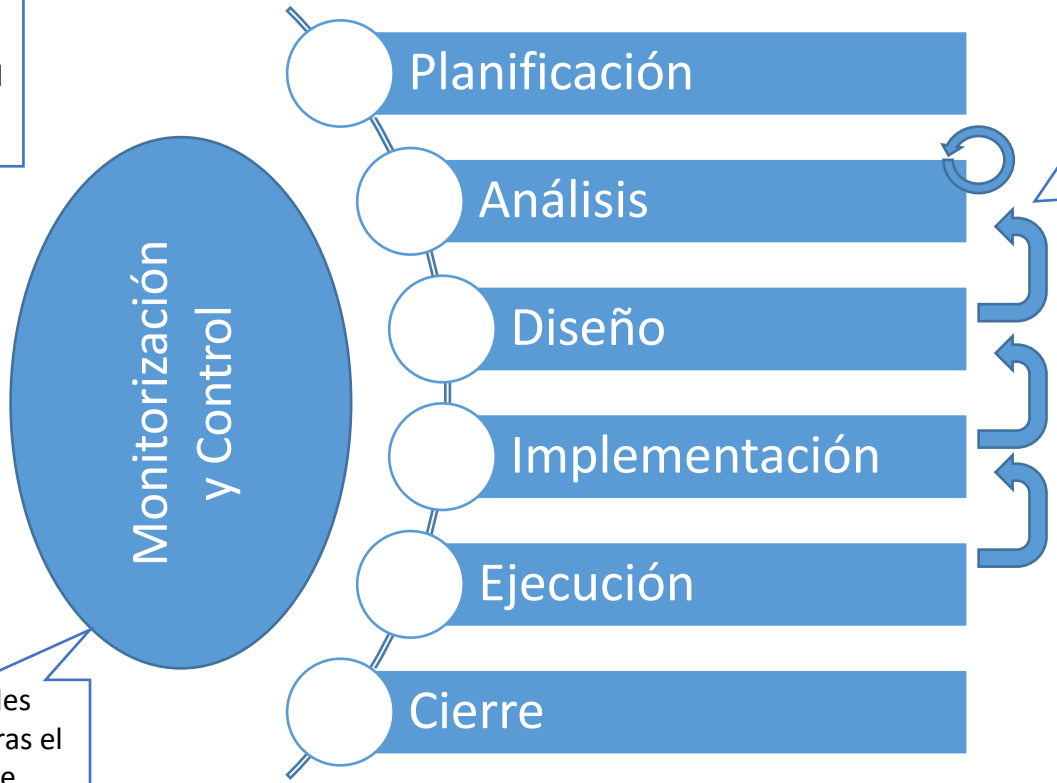
Recordemos que "El testing depende del contexto" (principio 6).
De todos modos, siempre podemos (y debemos) medir el proceso para saber cómo estamos y comparar.

Como parte del contexto, el testing se ve influido por:

- Estrategia, estándares y herramientas de cada organización
- Métodos usados para gestionar el software y su ciclo de vida
- Dominio del producto (no es lo mismo sw para un avión que para una ferretería)
- Restricciones (de proyecto, tiempo, esfuerzo, conocimiento, …)
- Marco legal y regulatorio
- La cultura y mentalidad de la organización
- … entre otros

Actividades del proceso de testing

La secuencia depende también del SDLC



Algunas actividades se pueden solapar o combinar

Actividades continuas tras el arranque

Referencias

- David Garvin (“Competing on the Eight Dimensions of Quality”) HBR

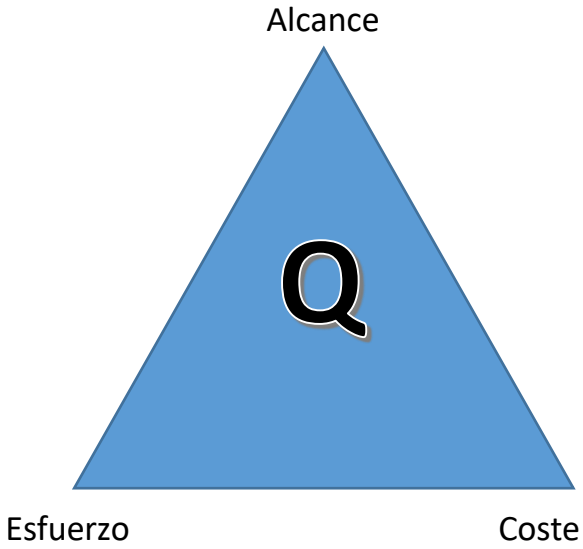
https://hbr.org/1987/11/competing-on-the-eight-dimensions-of-quality

- Boehm, B. y V. Basili, “Software Defect Reduction Top 10 List”, IEEE Computer
- Roger Pressman, “Ingeniería de software un enfoque práctico”
- Software Quality https://en.wikipedia.org/wiki/Software_quality
- Quality Assurance https://en.wikipedia.org/wiki/Quality_assurance
- Northeast blackout of 2003 https://en.wikipedia.org/wiki/Northeast_blackout_of_2003
- Sesgos cognititivos https://es.wikipedia.org/wiki/Anexo:Sesgos_cognitivos

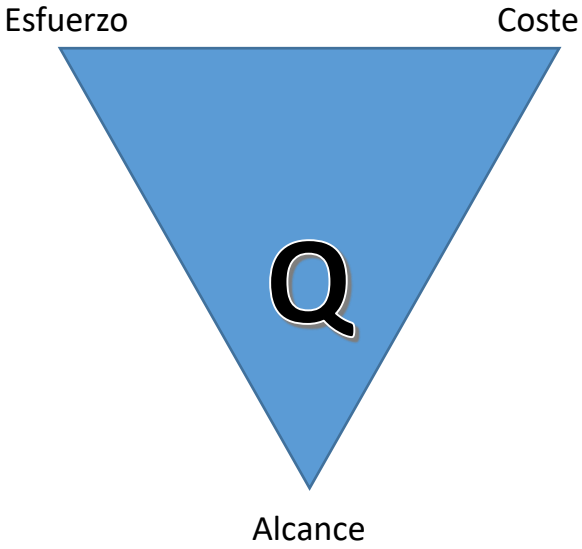
Anexo. Información adicional

¿Qué es Calidad?

Triángulos de hierro: ¿dónde ubicamos a la calidad?



Proyecto predictivo, tradicional o waterfall



Proyecto ágil (Scrum, Kanban, SAFe)

Dimensiones de la Calidad

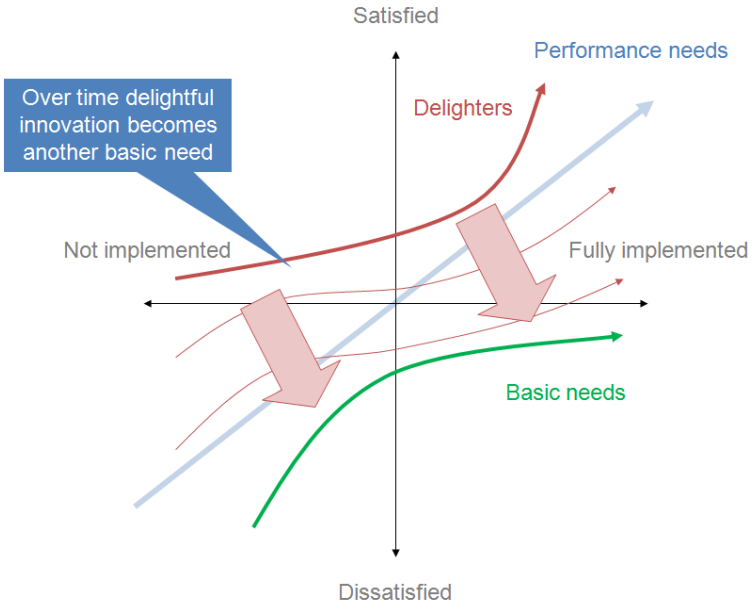
Estamos viendo que la calidad tiene muchas caras, esta es una descripción bastante completa:

- Calidad del desempeño: ¿se entregan todos los requisitos dando valor al usuario final?
- Calidad de las características: ¿sorprende y agrada la primera vez que lo emplean los usuarios finales?
- Confiabilidad: ¿proporciona todas las características y capacidades sin fallar?
- Conformidad: ¿concuerda con los estándares relevantes para la aplicación? ¿Y con el diseño implícito y las convenciones de código?
- Durabilidad: ¿se puede mantener y corregir sin efectos colaterales?
- Servicio: ¿se puede mantener en un tiempo y con un esfuerzo aceptable?
- Estética: solución "elegante"
- Percepción: transmite una sensación positiva. Es una forma de decir que "genera confianza".

David Garvin ("*Competing on the Eight Dimensions of Quality*") HBR
https://hbr.org/1987/11/competing-on-the-eight-dimensions-of-quality

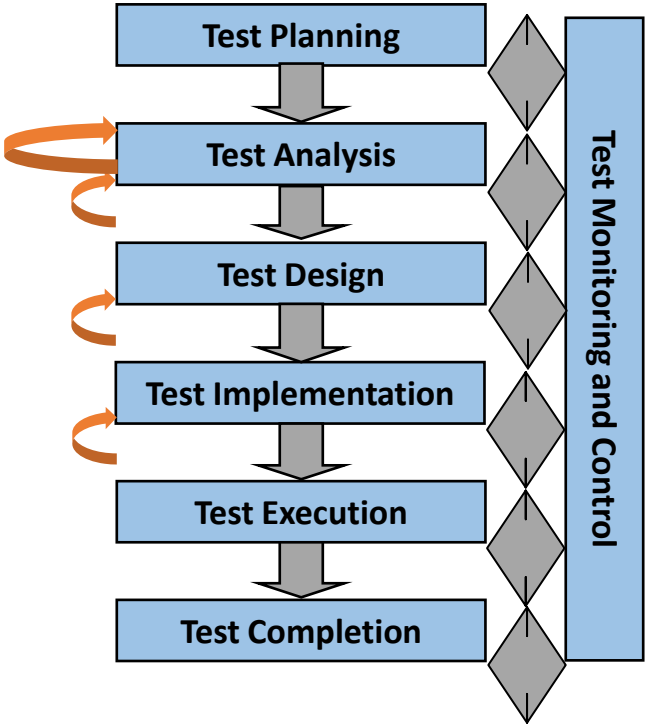
Percepción de la Calidad

La Calidad no es algo que sea percibido de la misma forma a lo largo del tiempo. Los atributos de cualquier producto o servicio que antes eran vistos como algo excepcional o inesperado pasan a ser algo habitual y su ausencia genera insatisfacción. La percepción evoluciona y se puede educar.



El modelo de Kano para el desarrollo de productos expresa esta idea de evolución de la percepción aplicable a la calidad.

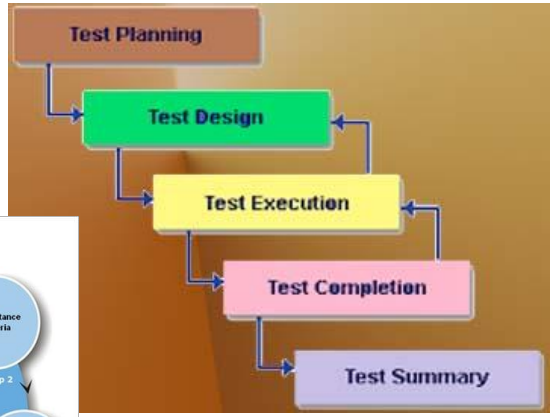
Actividades del proceso de testing

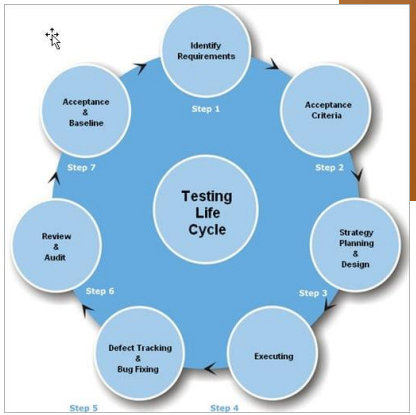


Modelo ISTQB

Hay más modelos

Pero comparten la mayor parte de las actividades y tareas





Planificación

Aquí es donde se determina:

- Los objetivos del proceso
- La forma de alcanzarlos:
  - Con qué actividades
  - Planificación temporal
  - Orientación
  - Restricciones
- Motivos (umbrales) para replanificar a partir de la información que proporciona la actividad de monitorización
- Resultados del proceso
  - Planes de pruebas (test plan) incluyendo todo lo necesario para la monitorización y el control (trazabilidad, fechas, criterios de finalización)

Monitorización y control

Monitorización: se trata de verificar el progreso y desviaciones del plan usando métricas que se han definido previamente.

Control: lanzar acciones para ajustarse a los objetivos del plan si se detectan desviaciones que los hagan peligrar.

Todo ello está dirigido por el análisis de los resultados para comprobar si se cumplen los criterios de finalización (o "*exit criteria*" o *Definition of Done*).
Básicamente se examinará el nivel de calidad alcanzado para determinar si es necesario realizar pruebas adicionales.

La monitorización y control genera informes de progreso, cobertura, resultados, …

Análisis

Esta es la etapa en la que se identifica qué elementos probar y cómo hacerlo:

- Funcionalidades de los elementos sujetos a prueba
- Condiciones para la prueba
- Definición de cobertura de test (aspectos probados sobre el conjunto)

Esto supone:

- Recopilar las bases de las pruebas (requisitos, diseños, información sobre la implementación, definiciones, el código del producto, riesgos, definiciones, …).
- Evaluarlas, lo que permitirá detectar defecto como omisiones, inconsistencias, ambigüedad, …
- Identificar qué probar
- Priorizar las condiciones de test
- Definir la trazabilidad (entre pruebas y elementos que las definen)

El resultado es:

- Condiciones de prueba priorizadas, de alto o bajo nivel

Diseño

Es el momento en el que se construyen los test cases o casos de prueba.

Esto supone:

- Diseñar, agrupar y priorizar test cases
- Identificar los datasets necesarios para los test cases
- Diseñar el entorno de pruebas, e identificar herramientas y elementos necesarios

Y como resultado tendremos:

- Tests cases y paquetes de pruebas
- Datasets
- Diseño de entornos de prueba

Implementación

Esta es la etapa en la que se implementan los test cases para convertir su definición en una serie de procedimientos, código, … que llevan a cabo la prueba.

Esto supone:

- Desarrollos los procedimientos de test (incluyen los automáticos)
- Crear conjuntos, paquetes o suites de tests
- Preparar los mecanismos para su ejecución
- Construir los entornos de prueba
- Extraer y preparar los datasets

A veces es una actividad realizada junto al diseño.

El resultado es:

- Procedimientos y scripts de prueba
- Paquetes de pruebas
- Datos de prueba
- Entornos de prueba
- Otros elementos necesarios para los tests

Además, es posible que haya cambios y actualizaciones sobre la definición de los tests.

Ejecución

Es normalmente la etapa que consume más tiempo y esfuerzo, sobre todo porque se repite con frecuencia (regresiones). Supone la ejecución de los casos de prueba.

Esto supone:

- Ejecución manual o automática
- Identificar diferencias entre los resultados esperados y los obtenidos
- Analizar esas diferencias para identificar fallos, e incluso el origen de esos fallos (defectos)
- Generar informes
- Validar la reparación (fix) de los defectos detectados

Como resultados tendremos:

- Informes (de ejecución, fallos, …)
- Feedback de mejora de los tests
- Registros y trazabilidad

Cierre

Acciones cuando se han alcanzado las condiciones de salida como completar la ejecución de todos los paquetes de pruebas, o haber alcanzado un nivel de calidad determinado, o haber alcanzado una fecha límite (no es un buen criterio), o finalizar la vigencia del objeto que se está probando (obsolescencia, sustitución, …).

Lo que supone:

- Generar aprendizajes: resúmenes, post-morten, lecciones aprendidas
- Documentar adecuadamente el proceso y sus resultados

Y los resultados suelen ser:

- Informes
- Acciones de mejora
- Documentación

Si el desarrollo exige rigor, el testing es aún más riguroso