Preguntas. Segundo Parcial

Unidad 4.1: Pruebas y estrategias de pruebas de software

1) Definición: Pruebas, datos de pruebas, caso de pruebas, depuración, equivocación, cobertura, refactorizar, condición de prueba, defecto, fallo y error, verificación y validación, estrategias de pruebas, depuración.

- Pruebas: Una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y registran y se realiza una evaluación de algún aspecto. Un conjunto de casos y procedimientos de prueba.

  El testing es una investigación técnica de un producto bajo prueba con el fin de brindar información relativa a la calidad del software, a los diferentes actores involucrados en un proyecto. Es una actividad cognitiva y no mecánica ni repetitiva que involucra a varias funciones mentales, como el lenguaje, la imaginación percepción entre otros.

- Datos de Pruebas: Lotes de datos necesarios para ejecutar un caso de test

- Casos de Pruebas: Un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular. También se puede referir a la documentación en la que se describen las entradas, condiciones y salidas de un caso de prueba.

- Depuración: el proceso de localizar, analizar y corregir los defectos que se sospecha que contiene el software.

- Equivocación (mistake): Una acción del ser humano que produce un resultado incorrecto.

- Cobertura: Es una medida de que tan completo fue el testing (en función da una estrategia particular) Toda estrategia tiene asociado su concepto de cobertura.

- Refactorizar: Es la modificación de la estructura interna de un programa con la finalidad de obtener mejor legibilidad y escalabilidad, de tal forma que el comportamiento observable del software modificado no se vea afectado.

- Condición de prueba: Elemento o evento de un componente o sistema que debería ser verificado por uno o más casos de prueba, por ejemplo una función, transacción, característica, atributo de calidad o elemento estructural.
- Defecto(defect, fault, bug): Un defecto en el software, como por ejemplo un proceso, es una definición de datos o un paso de procesamiento incorrectos en un programa.

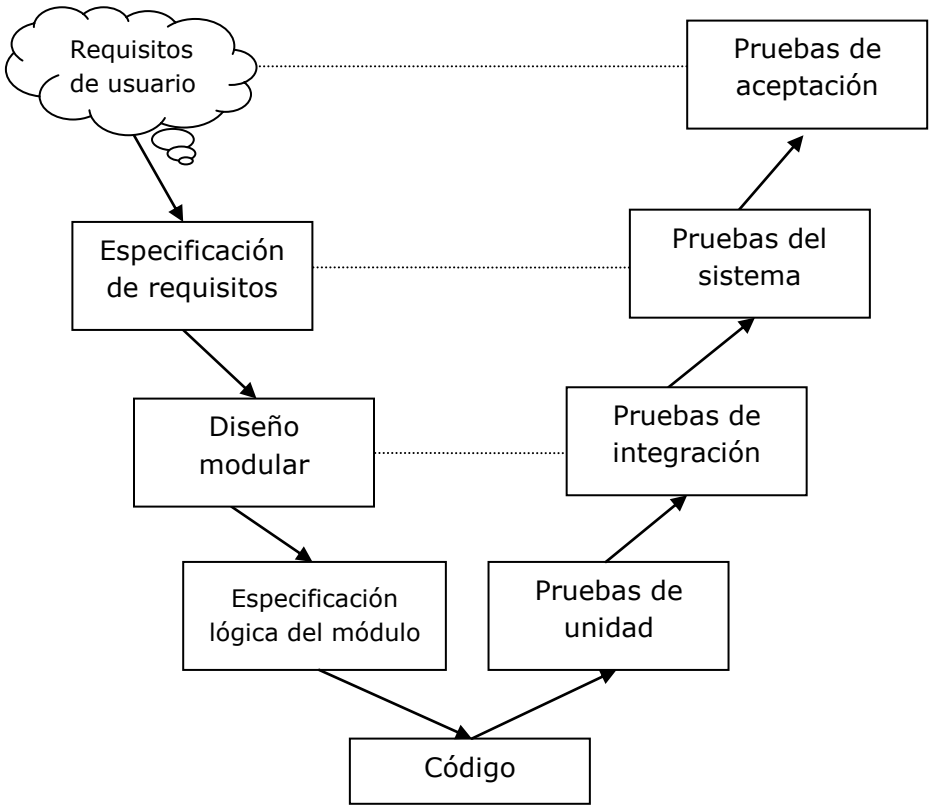
- Fallo(failure): La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados

- Error: tiene varias aceptaciones-la diferencia entre un valor calculado, observado o medio y el valor verdadero, especificado o teóricamente correcto.

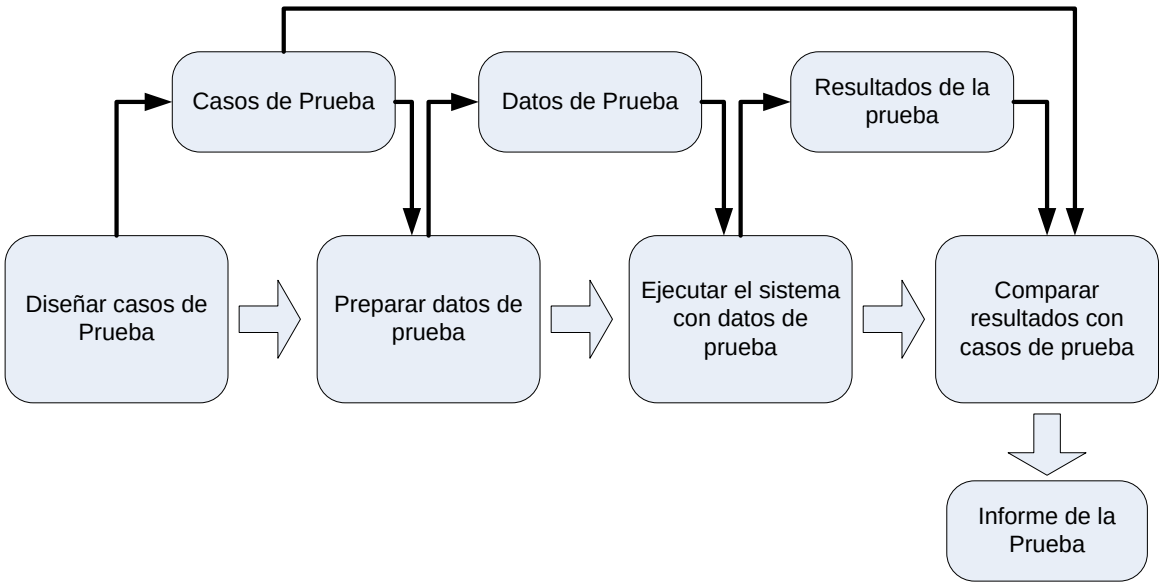
- Verificación: El proceso de evaluación de un sistema o de uno de sus componentes para determinar si los productos de una fase dada satisfacen las condiciones impuestas al comienzo de dicha fase. La verificación se realiza "contra" el entorno de desarrollo o del proyecto. La verificación sirve para saber si se ha construido el producto correctamente.

- Validación: El proceso de evaluación de un sistema o de uno de sus componentes durante o al final del proceso de desarrollo para determinar si satisface los requisitos marcados por el usuario. La validación se realiza "contra" el entorno del cliente, donde el producto debe cumplir su finalidad. La validación es para saber si se ha construido el producto correcto.

2) Relación entre producto de desarrollo y niveles de pruebas: Modelos en V del ciclo de vida.



3) Proceso de pruebas y sus entregables.



Una vez que se ha generado el código, comienzan las pruebas del programa. El proceso de pruebas se centra en los procesos lógicos internos del software, asegurando que todas las sentencias se han comprobado, y en los procesos externos funcionales; es decir, realizar las pruebas para la detección de errores y asegurar que la entrada definida produce resultados reales de acuerdo con los resultados requeridos.

Entregables:

- El plan de prueba: describe todos los métodos que se utilizarán para verificar que el software satisface la especificación del producto y las necesidades del cliente. Incluye los objetivos de calidad, necesidades de recursos, cronograma, asignaciones, métodos, etc.
  Es el documento que describe el alcance, enfoque, los recursos y planificación de las actividades de pruebas previstas. Identifica, entre otros, los elementos de prueba, las prestaciones a ser probadas, las tareas de pruebas, quien realiza cada tarea, el grado de independencia del probador, el entorno de pruebas, las técnicas de diseño de pruebas y los criterios de entrada y salida a utilizar, y los motivos para cada elección, y cualquier riesgo que requiera un plan de contingencia.
  Es un registro del proceso de planificación de pruebas. [Según IEEE 829]

- Casos de prueba: lista los ítems específicos que serán probados y describe los pasos detallados que serán seguidos para verificar el software.

- Reporte de bugs: describen los problemas encontrados al ejecutar los casos de prueba.

- Herramientas de pruebas y automatización: documentación de las herramientas empleadas en el proceso de pruebas.

- Métricas, estadísticas y resúmenes: indican como ha sido el progreso del proceso de prueba. Toman la forma de gráficos, tablas e informes escritos.

De acuerdo con el estándar IEEE Std. 829, los documentos relacionados con el diseño de pruebas son:

- Plan de pruebas
- Especificación de diseño de pruebas
- Especificación de casos de prueba
- Especificación de procedimientos de pruebas

Y los documentos relacionados con la ejecución de pruebas son:

- Histórico de pruebas (test log)
- Informe de incidentes
- Informe resumen de las pruebas

4) Roles, responsabilidades y perfil de tester

Roles:

A lo largo de estos últimos años se ha visto más evidente la necesidad de incluir a equipos que se dediquen a hacer la validación de productos de software, los que comúnmente conocemos como "testers", de igual forma se ha podido relacionar el grado de éxito de un proyecto con el software testing.

- ***Probadores o personal QA - Aseguradores de la calidad (Testers or QA (Quality Assurance) Staff)***: son responsables de encontrar y reportar los problemas en el producto de software. Trabajan conjuntamente con todos los miembros del equipo informándoles como se está desarrollando y ejecutando las pruebas y reportando los problemas encontrados.
    - Rol del tester:
        - Investiga un producto de software con el objetivo de obtener información acerca de su calidad y del valor que representa para quienes lo utilizan.
        - Asume el desafío de detectar la mayor cantidad de fallas severas (incidentes de alto impacto) con el mínimo esfuerzo, antes de que el software salga a producción.
        - Participa de todas las etapas del proceso de desarrollo de software, colaborando para asegurar la máxima calidad del producto. Su perfil conjuga un conjunto de habilidades con el conocimiento del negocio, de la aplicación bajo prueba y de cómo planificar, diseñar, ejecutar y administrar las pruebas.
    - Responsabilidades del Tester:
        - Identificar las pruebas que se requiere llevar a cabo
        - Construir un plan de testing
        - Coordinar con los diseñadores para incluir el test del diseño en el documento

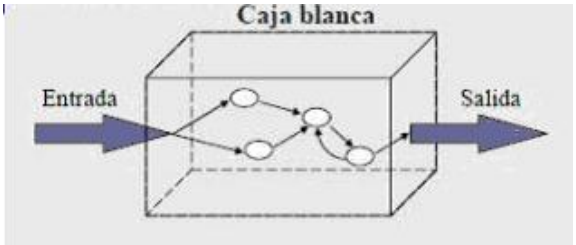
- Realizar I £D de las herramientas y arquitecturas
- Identificar el acercamiento más apropiado para implementar una prueba dada
- Preparar y ejecutar las pruebas
- Registrar resultados y verificar que las pruebas hayan sido ejecutadas
- Análisis y recuperación de errores de ejecución.
- Comunicar los resultados de las pruebas al equipo
- Participar en la revisión de los requisitos del sistema
- Construir la documentación del proceso de testing con sus métricas.

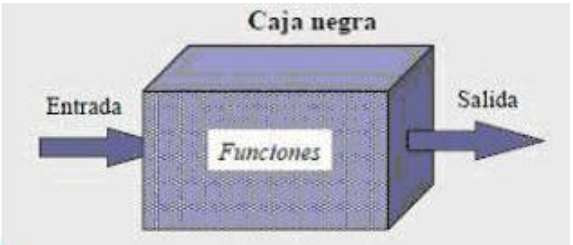
-
- Perfil de un Testing:
    - Capacidad de abstracción y modelado para entender y simular el comportamiento del sistema bajo prueba.
    - Curiosidad e intuición, explorar situaciones sonde se pueden encontrar errores
    - Facilidad de comunicación oral y escrita para interactuar con desarrolladores y usuarios.
        - Comunicar de forma clara y precisa los defectos encontrados y discutirlos.
    - Capacidad de actuar bajo presión
        - Actuar cuando la presión del plazo es mayor
        - Manejo de conflicto de intereses con el equipo de desarrollo.
    - Creatividad para generar ideas e imaginar los problemas que podrían existir.
    - Pensamiento crítico para evaluar las ideas, hacer deducciones y vincular lo observado con los criterios de calidad de la empresa.
    - Pragmatismo para poner en práctica las ideas y adecuar las técnicas y el esfuerzo al alcance del proyecto.
    - Aptitudes para el trabajo en equipo, de manera de poder interactuar con los desarrolladores y otros testers, y lograr el máximo beneficio en esta interacción.

Estas habilidades se adquieren y perfeccionan a medida que se avanza en la carrera de testing y se obtiene conocimiento y experiencia.

5) Objetivos de la pruebas de caja blanca y negra. Criterios de aceptación y variantes de caja blanca y caja negra.

Las pruebas de caja blanca necesitan conocer los detalles procedimentales del código, mientras que las de caja negra únicamente necesitan saber el objetivo o funcionalidad que el código ha de proporcionar.





En realidad estos dos tipos de técnicas son técnicas complementarias que han de aplicarse al realizar una prueba dinámica, ya que pueden ayudar a identificar distintos tipos de faltas en un programa

- Pruebas de caja blanca:

  Conociendo el funcionamiento del producto, se desarrollan pruebas que aseguren "que todas las piezas encajan", que la operación interna se ajusta a las especificaciones y que todos los componentes internos se han comprobado adecuadamente. Es un examen minucioso de detalles procedimentales

  - *Objetivos*
    - También llamadas pruebas de caja transparente son aquellas que se concentran en el código fuente.
    - No se puede tener una prueba que modele el 100% de todos los casos de uso del sistema. Se debe realizar una prueba de segmentos. Un segmento es un bloque de instrucción
    - Garantizar que se ejerciten por lo menos una vez todos los caminos independientes de cada módulo
    - Ejercitar todas las decisiones lógicas en sus vertientes verdadera y falsa
    - Ejecutar todos los bucles en sus límites
    - Ejercitar las estructuras de datos para asegurar su valide
- *Criterios de cobertura Lógica* (de Menos rigurosos (Más baratos) a Mas rigurosos (Mas caros))
  - Cobertura de Sentencias
  - Cobertura de Decisiones
  - Cobertura de Condiciones
  - Criterios de decisión/Condición
  - Criterios de Condición Múltiple
  - Criterio de cobertura de Caminos (impracticable)
- *Variantes de Prueba de Caja Blanca*
  a) Prueba del camino Básico: complejidad ciclomatica de McCabe, es un indicador del numero de caminos independientes que existen en un grafo.
  b) Prueba de Condición: Da una orientación para generar pruebas adicionales del programa
  c) Pruebas de Flujo de Datos: Se seleccionan caminos de un programa de acuerdo a las definiciones y uso de las variables.
  d) Prueba de Bucles: Es según los tipos de bucles (simples, anidados, concatenados, no estructurados)

Pruebas de Caja Negra:

Conociendo la función para la que fue diseñado, se hacen pruebas que demuestren que cada función es operativa y al mismo tiempo se buscan errores en cada una. Son pruebas sobre la interfaz del software.

Están orientadas a lo que se espera realicen las parte modulares del software.

Son pruebas funcionales y no interesa como se realiza la tarea

Son recomendables realizarlas con los usuarios

Se centran en los requisitos funcionales del software. No es una alternativa a las técnicas de caja blanca, más bien se trata de un enfoque complementario que intenta descubrir errores de las siguientes categorías:

- Funciones incorrectas o ausentes
- Errores de interfaz
- Errores en la estructura de datos o en accesos a bases de datos externas
- Errores de rendimiento
- Errores de inicialización y de terminación

Variantes de Pruebas de caja negra:

a) Métodos de prueba basados en grafos
b) Partición Equivalente:
c) Análisis de valores limite
d) Prueba de comparación
e) Conjetura de errores

6) ¿Qué es y qué incluye un plan de pruebas?

Un plan de pruebas traza la clase de pruebas que se han de llevar a cabo. Un procedimiento de pruebas define los casos de prueba específicos para descubrir los errores de acuerdo con los requisitos

Tanto el plan como el procedimiento estarán diseñados para asegurar que se satisfacen todos los requisitos funcionales, que se almacenan todos los requisitos de rendimiento, que la documentación es correcto o legible y que alcanzan otros requisitos (por ejemplo portabilidad, compatibilidad, recuperación de errores, facilidad de mantenimiento)

El propósito real del plan de pruebas consiste en explicar el alcance, enfoque, recursos requeridos, calendario, responsables y el manejo de riesgos de un proceso de pruebas.

Incluye:

1. Identificador del plan
   Preferiblemente de alguna forma mnemónica que permita relacionarlo con su alcance, por ej. TP-Global (plan global del proceso de pruebas), TP-Req-Seguridad1 (plan de verificación del requerimiento 1 de seguridad), TP-Contr-X (plan de verificación del contrato asociado al evento de sistema X), TP-Unit-Despachador.iniciar (plan de prueba unitario para el método iniciar de la clase Despachador). Como todo artefacto del desarrollo, está sujeto a control de configuración, por lo que debe distinguirse adicionalmente la versión y fecha del plan.

2. Alcance
   Indica el tipo de prueba y las propiedades/elementos del software a ser probado.

3. Items a probar
   Indica la configuración a probar y las condiciones mínimas que debe cumplir para comenzar a aplicarle el plan. Por un lado, es difícil y riesgoso probar una configuración que aún reporta fallas; por otro lado, si esperamos a que todos los módulos estén perfectos, puede que detectemos fallas graves demasiado tarde.

4. Estrategia
   Describe la técnica, patrón y/o herramientas a utilizarse en el diseño de los casos de prueba. Por ejemplo, en el caso de pruebas unitarias de un procedimiento, esta sección podría indicar: "Se aplicará la estrategia caja-negra de fronteras de la precondición" o "Ejercicio de los caminos ciclomáticos válidos". En lo posible la estrategia debe precisar el número mínimo de casos de prueba a diseñar, por ej. 100% de las fronteras, 60% de los caminos ciclomáticos. La estrategia también explicita el grado de automatización que se exigirá, tanto para la generación de casos de prueba como para su ejecución.

5. Categorización de la configuración
   Explicita las condiciones bajo las cuales, el plan debe ser:
   1. Suspendido,
   2. Repetido;
   3. Culminado.

   En algunas circunstancias (las cuales deben ser explicitadas) el proceso de prueba debe suspenderse en vista de los defectos o fallas que se han detectado. Al corregirse los defectos, el proceso de prueba previsto por el plan puede continuar, pero debe explicitarse a partir de qué punto, ya que puede ser necesario repetir algunas pruebas. Los criterios de culminación pueden ser tan simples como aprobar el número mínimo de casos de prueba diseñados o tan complejos como tomar en cuenta no sólo el número mínimo, sino también el tiempo previsto para las pruebas y la tasa de detección de fallas.

6. Tangibles
   Explicita los documentos a entregarse al culminar el proceso previsto por el plan p. ej. subplanes, especificación de pruebas, casos de prueba, resumen gerencial del proceso y bitácora de pruebas.

7. Procedimientos especiales
   Identifica el grafo de las tareas necesarias para preparar y ejecutar las pruebas, así como cualquier habilidad especial que se requiere.

8. Recursos
   Especifica las propiedades necesarias y deseables del ambiente de prueba, incluyendo las características del hardware, el software de sistemas (p. ej. el sistema de operación), cualquier otro software necesario para llevar a cabo las pruebas, así como la colocación específica del software a probar (p. ej. qué módulos se colocan en qué máquinas de una red local) y la configuración del software de apoyo.

   La sección incluye un estimado de los recursos humanos necesarios para el proceso. También se indican cualquier requerimiento especial del proceso: actualización de licencias, espacio de oficina, tiempo en la máquina de producción, seguridad.

9. Calendario
   Esta sección describe los hitos del proceso de prueba y el grafo de dependencia en el tiempo de las tareas a realizar.

10. Manejo de riesgos
    Explicita los riesgos del plan, las acciones mitigantes y de continencia.

11. Responsables
    Especifica quién es el responsable de cada una de las tareas previstas en el plan.

7) Pruebas unitarias, de integración, de validación, de defectos, del sistema, de aceptación, de interfaz usuario, de regresión, exploratorias y ad-hoc, alfa y beta.

- Pruebas Unitarias:
  La prueba de unidad es la primera fase de las pruebas dinámicas y se realizan sobre cada módulo del software de manera independiente. El objetivo es comprobar que el módulo, entendido como una unidad funcional de un programa independiente, está correctamente codificado.
  Centra el proceso de verificación en la menor unidad del diseño del software: el modulo. Usando la descripción del diseño procedimental como guía, se prueban los caminos de control importantes, con el fin de descubrir errores dentro del límite del módulo.

- Pruebas de Integración: La prueba de integración es una técnica sistemática para construir la estructura del programa mientras al mismo tiempo, se lleva a cabo pruebas para detectar errores asociados con la interacción. El objetivo es tomar los módulos probados en unidad y estructurar un programa que esté de acuerdo con el que dicta el diseño
  Usa fundamentalmente técnicas de caja negra. Algunas veces usa técnicas de prueba de caja blanca para asegurar que se cubre los principales caminos de control

- Pruebas de Validación:

Intentan demostrar que el software es el que el cliente quiere (que satisfaga sus requerimientos). Como parte de la prueba de validación, se pueden utilizar pruebas estadísticas para probar el rendimiento y la fiabilidad de los programas, y para comprobar cómo trabaja en ciertas condiciones operacionales.
El objetivo es comprobar que el sistema se desempeñe correctamente utilizando casos de pruebas dados.

- Pruebas de defectos: intentan revelar defectos en el sistema en lugar de simular su uso operacional. El objetivo de las pruebas de defectos es hallar inconsistencias entre un programa y su especificación, exponiendo los defectos latentes en un sistema de software antes de entregar el sistema.
  Durante esta prueba algunas mostraran que el programa satisface sus requerimientos.

- Pruebas de Sistema: Este tipo de pruebas tiene como propósito ejercitar profundamente el sistema para verificar que se han integrado adecuadamente todos los elementos del sistema (hardware, otro software, etc.) y que realizan las funciones adecuadas. Concretamente se debe comprobar que:
  - Se cumplen los requisitos funcionales establecidos.
  - El funcionamiento y rendimiento de las interfaces hardware, software y de usuario.
  - La adecuación de la documentación de usuario.
  - Rendimiento y respuesta en condiciones límite y de sobrecarga.

  Para la generación de casos de prueba de sistema se utilizan técnicas de caja negra.
  Este tipo de pruebas se suelen hacer inicialmente en el entrono del desarrollador, denominadas Pruebas Alfa, y seguidamente en el entrono del cliente denominadas Pruebas Beta.

- Pruebas de aceptación: Pruebas formales con respecto a las necesidades de usuario, requisitos y procesos de negocio dirigidas a determinar si el sistema satisface o no los criterios de aceptación y a habilitar al usuario, cliente u otra entidad autorizada a determinar si acepta o no el sistema. [Según IEEE 610]
  Es necesario asegurar:
  - Que las pruebas se ejecutan
  - Que los resultados se publican
  - Que se establecen líneas base
  - Podemos extraer métricas de los resultados de las pruebas manuales

- Pruebas de interfaz usuario: La prueba de interfaz de usuario verifica la interacción del usuario con el software. El objetivo es asegurar que la interfaz tiene apropiada navegación a través de las diferentes funcionalidades.
  Estrategia viable de pruebas de interfaz

  La estrategia global para las pruebas de interfaz es:
  Descubrir errores relativos a los mecanismos específicos de cada interfaz.
  Descubrir errores en la forma en la que la interfaz implementa la semántica de la navegación.

Probar mecanismos específicos de una interfaz

Los usuarios interactúan con la interfaz por medio de una serie de mecanismos que serán los que tendremos que probar:

- Enlaces
- Formularios
- Scripts del lado del cliente
- HTML dinámico
- Ventanas emergentes
- Scripts del lado del servidor
- Contenido embebido
- Cookies
- Otros mecanismos específicos de la interfaz

- Pruebas de regresión: Las pruebas de regresión consisten en volver a ejecutar un subconjunto de las pruebas realizadas anteriormente, con el fin de asegurarse de que los cambios no han propagado efectos laterales no deseados

  Fuera de la prueba de integración, las pruebas de regresión también se utilizan cuando cambia la configuración del software (programa, documentación o datos) ⇒ p.ej. durante el mantenimiento correctivo

- Pruebas exploratorias: Las pruebas exploratorias muestran sus resultados y garantías en base a los resultados y garantías de prácticas previas y del mismo modo, fallan sus resultados y garantías cuando prácticas que le anteceden son ejecutadas en forma endeble o imprecisa.

  Para esta práctica son necesarios ciertos elementos que le anteceden:

  - Conciencia de que todo lo que se verifica está orientada a la detección de problemas que impidan satisfacer requisitos funcionales y no funcionales
  - Capacidad de análisis elevado para entendimiento cabal de requisitos en base a la observación y análisis de la evolución de los mismos
  - Abstracción para la elaboración de circuitos de pruebas de verificación y validación
  - Capacidad de ampliación de caminos de pruebas para mejora espontánea de la cobertura
  - Conocimiento y dominio del negocio al que pertenece el problema
  - Alta proactividad y gran capacidad comunicacional para generar ambiente colaborativo
  - Conocimiento cabal del proceso que rige para reconocer las fallas que se manifiestan en los resultados no deseados
  - Gran capacidad para definir y variar las estrategias, tácticas y técnicas de validación y verificación
  - Olfato e intuición amplificada

- Testing aleatorio ad-hoc: Pruebas llevadas a cabo de manera informal; no se realiza una preparación formal de la prueba, no se utilizan técnicas de diseño reconocidas, no existen expectativas para con los resultados y la arbitrariedad guía la actividad de ejecución

- Pruebas alfa: Se hace en el lugar de desarrollo y con un cliente que la realiza. El desarrollador en un observador del usuario y registra errores y problemas de uso.

- Pruebas beta: La hacen los usuarios finales. Se hacen después de haber admitido las pruebas del sistema y las pruebas alfa.

8) Pruebas de aplicación basadas en Web: Que son, Cual es su producto final y estrategias para probar.

Las pruebas de webapp son una colección de actividades relacionadas con una sola meta: descubrir errores en el contenido, función, utilidad, navegabilidad, rendimiento, capacidad y seguridad de esa aplicación.

Para lograr esto se aplica una estrategia de prueba que abarca tanto revisiones como pruebas ejecutables

¿Cuál es el producto final?

- Plan de pruebas para la webapp
- Suite de casos de prueba para cada paso de prueba y se mantiene un archivo de los resultados de la prueba para un uso futuro.

Estrategias para probar Webapp

Pasos que resumen el enfoque para hallar errores:

1. Se revisa el modelo de contenido
2. Se examina el modelo de interfaz para garantizar que todos los casos de uso pueden alojarse.
3. El modelo de diseño para la weapp se revisa para descubrir errores de navegación.
4. Se prueba la interfaz de usuario se prueba para descubrir errores en los mecanismos de presentación y/o navegación.
5. A cada componente funcional se le aplica una prueba de unidad unidad.
6. Se prueba la navegación a lo largo de toda la arquitectura.
7. La webapp se implementa en varias configuraciones ambientales diferentes y se prueba en su compatibilidad con cada configuración.
8. Las pruebas de seguridad se realizan con la intención de explotar vulnerabilidades en la webapp o dentro de su entorno.
9. Se realizan pruebas de rendimiento
10. La webapp se prueba mediante una población de usuarios finales controlada y monitoreada. Los resultados de su interacción con el sistema se evalúan por errores de contenido y navegación,

preocupaciones de facilidad de uso, preocupaciones de compatibilidad, así como confiabilidad y rendimiento de la webapp.

9) Herramientas de soporte para la gestión y para la especificación de pruebas: cuáles son y cuál es su objetivo

Herramientas de gestión de pruebas

- Soporte para la gestión de pruebas y el cumplimiento de las actividades de prueba.
- Interfaces para las herramientas de ejecución de test, herramientas de traza de defectos y herramientas de gestión de requerimientos.
- Control de versión independiente o una interfaz con una herramienta de gestión de configuración externa.
- Soporte para la trazabilidad de las pruebas, resultado e incidentes de los test para documentos fuentes, tales como especificación de requisitos.
- (Logging) Registro de resultados de tests y generación de informes de progreso.
- Análisis cuantitativo (métrica) relacionado con las pruebas (ej. test ejecutados test pasados) y el objeto de los test (ej. incidentes ocurridos), para dar información sobre el objeto del test, y controlar y mejorar el proceso de prueba.

Herramienta de gestión de requerimientos

- Almacenan la declaración de requerimientos
- Comprueban la consistencia y los requerimientos no definidos (perdidos)
- Permiten priorizar los requisitos
- Permite en test individuales rastrear los requerimientos, las funciones y/o las características.
- La trazabilidad puede ser presentada en informes de gestión en progreso de las pruebas.
- La cobertura de requerimientos, funciones y/o características para un conjunto de pruebas pueden ser también informados.

Herramienta de gestión de incidentes

También conocidos como herramientas de seguimiento de defectos (defect tracking tools)

- Almacenan y administran los informes de incidentes (defectos, errores o anomalías)
- Apoyan la gestión de los informes de incidentes de la siguiente forma:
    - Facilitando su priorización
    - Asignación de acciones a las personas (fijar o confirmar tests)
    - Atribución de estados (por ej.: rechazados, preparados para ser comprobados, plazo para una próxima publicación.)
- Permiten que el proceso de incidentes sea monitorizado a lo largo del tiempo,
- Proporcionan apoyo para análisis estadístico
- Proporcionan informes sobre los incidentes.

Herramientas de gestión de configuración

No son estrictamente herramientas de prueba, pero son necesarias para seguir el rastro de las diferentes versiones y desarrollos de software y pruebas.

- Almacenan información sobre versiones y desarrollos de software y testware.
- Permiten la trazabilidad entre los productos de testware y productos de trabajo software y variantes de productos.
- Son particularmente útiles cuando se desarrolla para más de una configuración de entornos hardware/software (ej.: para diferentes versiones de sistemas operativos, diferentes librerías o compiladores, diferentes buscadores o computadoras.)

Herramientas de diseño de pruebas

- Generan entradas de tests o verdaderos tests de requisitos, de interfaz gráfico de usuario, de modelos de diseños (estado, datos u objeto) o de código.
- El test generado es útil para verificar la implementación del modelo del software o sistema.
- Pueden ahorrar un tiempo valioso y proporcionar un incremento de la minuciosidad de las pruebas como consecuencia de la complejidad del test que la herramienta puede generar.

Herramientas de preparación de las pruebas de datos

- Manipulan la base de datos, ficheros o transmisión de datos para configurar una prueba de datos para ser usado durante la ejecución de los test.
- Aseguran que la transferencia de datos a un entorno de test sea hecho anónimamente, para la protección de datos.

Herramientas de ejecución de tests

- Ejecutan scripts (diseñados para implementar las pruebas)
- Script de captura: representación lineal con datos específicos y acciones como parte de cada script.
- Herramientas de pruebas de funcionamiento
- Necesitan a alguien con experiencia en las pruebas de funcionamiento para ayudar a diseñar los test e interpretar los resultados

10) Automatización de pruebas: cómo se hace, beneficios y riesgos

Las herramientas de prueba pueden:

- Mejorar la eficiencia de las actividades de prueba mediante la automatización de tareas repetitivas.
- Mejorar la fiabilidad de las pruebas mediante, por ejemplo, la automatización de la comparación de numerosos datos o la simulación de comportamientos.

¿Cómo se automatiza?

- En la especificación del procedimiento de pruebas, se ponen los casos de prueba en orden de ejecución.
- El procedimiento de pruebas (o manual del script de pruebas) especifica la secuencia de acciones para la ejecución de una prueba.

- Si las pruebas se ejecutan usando alguna herramienta de ejecución de pruebas, la secuencia de acciones se especifica en un script de pruebas (que es un procedimiento de pruebas automatizado).
- Las distintas pruebas y scripts de automatización de pruebas se estructuran dentro de una secuencia de ejecución de pruebas que define el orden en que los distintos procedimientos de pruebas, y los posibles scripts de automatización, son ejecutados, cuando son llevados a cabo por alguien.
- La secuencia de ejecución de pruebas tendrá en cuenta factores tales como pruebas de regresión, priorización y dependencias técnicas y lógicas.

Beneficios potenciales de usar herramientas incluyen:

- Los trabajos repetitivos se reducen (ej.: ejecución de pruebas regresivas, poniendo de nuevo los mismos datos del test, y comprobando otra vez estándares de códigos.)
- Mayor consistencia y capacidad de repetición (ej.: tests ejecutados por una herramienta, y test que provienen de requisitos)
- Valoración de objetivos (ej.: medidas estáticas, cobertura y comportamiento del sistema).
- Facilidad de acceso a información sobre los tests o pruebas (ej.: estadísticas y gráficos sobre el progreso de las pruebas, tasas de incidencias y ejecuciones.)

Los riesgos de usar herramientas incluyen:

- Expectativas irreales para la herramienta (incluyendo funcionalidad y facilidad de uso.)
- Infraestimación del tiempo, el coste y esfuerzo para la introducción inicial de la herramienta (incluyendo entrenamiento y experiencia externa)
- Infraestimación del tiempo y esfuerzo necesarios para conseguir beneficios significativos y continuos de la herramienta (incluyendo la necesidad de cambios en el proceso de pruebas y continua mejora de la forma en que se usa la herramienta.)
- Infra-estimación del esfuerzo requerido para mantener los recursos de las pruebas generadas por la herramienta.
- Sobredependencia de la herramienta (sustitución del diseño del test o donde las pruebas manuales serían mejor.)

Unidad 4.2: Testing ágiles

1) Definición de incremento

El incremento es la parte de producto producida en un sprint, y tiene como características que está completamente terminada y operativa: en condiciones de ser entregada al cliente final. No se trata por tanto de módulos o partes a falta de pruebas, o documentación. Idealmente en el desarrollo ágil:

- Cada funcionalidad del product backlog se refiere a funcionalidades entregables, no a trabajos internas del tipo “diseño de la base de datos”
- Se produce un “incremento” en cada iteración.

2) Esquema del desarrollo ágil: fases

1. Concepto

Se crea la visión del producto y se determina el equipo que lo llevará a cabo. Se necesita tener el concepto de lo que se quiere, y conocer el alcance del proyecto.

2. Especulación

Una vez que se sabe qué hay que construir, el equipo especula y formula hipótesis basadas en la información de la visión.

Se determinan las limitaciones impuestas por el entorno de negocio: costes y agendas principalmente, y se cierra la primera aproximación de lo que se puede producir.

3. Exploración

Se desarrolla un incremento del producto, que incluye las funcionalidades determinadas en la fase anterior

4. Revisión

Equipo y usuarios revisan lo construido hasta ese momento. Trabajan y operan con el producto real contrastando su alineación con el objetivo

5. Cierre

Al llegar a la fecha de entrega de una versión de producto (fijada en la fase de concepto y revisada en las diferentes fases de especulación), se obtiene el producto esperado. Posiblemente éste seguirá en el mercado, y por emplear gestión ágil, es presumible que se trata de un producto que necesita versiones y mejoras frecuentes para no quedar obsoleto. El cierre no implica el fin del proyecto.

Lo que se denomina “mantenimiento” supondrá la continuidad del proyecto en ciclos incrementales hacia la siguiente versión para ir acercándose a la visión del producto

3) Principios ágiles

- La gestión ágil se basa en los principios del manifiesto ágil y centra el valor:
- Más en las personas y su interacción que en los procesos y las herramientas.
- Más en los resultados que funcionan que en la documentación exhaustiva.
- Más en la colaboración con el cliente que en la negociación contractual.
- Más en la capacidad de respuesta al cambio que en el seguimiento de un plan.

4) Scrum: características distintivas, artefactos o elementos del modelo, reuniones, valores, reportes

Rasgos característicos:

- El trabajo de desarrollo se particiona en “paquetes”.
- Conforme se construye el producto se realizan las pruebas y la documentación.
- El trabajo ocurre en “carreras cortas” y se deriva de un “trabajo atrasado” de requerimientos existentes.
- Las reuniones son muy cortas y varias veces conducidas sin dirigentes.

- Los demos son entregados al cliente con el periodo de tiempo asignado.

Roles:

- Product Owner:
  - Define las funcionalidades del producto
  - Decide las fechas de lanzamiento y contenido
  - Acepta o no el trabajo realizado
- Scrum Master:
  - Responsable de la aplicación de los valores y prácticas de Scrum
  - Elimina impedimentos
  - Asegura que el equipo es completamente funcional y productivo
  - Garantiza la cooperación de los diferentes roles y funciones
- Team:
  - Entre 5-9 personas
  - Multi-funcional (programadores, diseñadores, testeadores)

Elementos:

- Pila del producto: (product backlog) lista de requisitos de usuario que a partir de la visión inicial del producto crece y evoluciona durante el desarrollo.
  Es recomendable el formato de lista que incluya al menos la siguiente información para cada línea:
  - Identificador único de la funcionalidad o trabajo.
  - Descripción de la funcionalidad.
  - Campo o sistema de priorización.
  - Estimación

  Dependiendo del tipo de proyecto, funcionamiento del equipo y la organización, pueden resultar aconsejables otros campos:
  - Observaciones
  - Criterio de validación
  - Persona asignada
  - Nº de Sprint en el que se realiza
  - Módulo del sistema al que pertenece

- Pila del sprint: (sprint backlog) lista de los trabajos que debe realizar el equipo durante el sprint para generar el incremento previsto.

  Condiciones

  - Realizado de forma conjunta por todos los miembros del equipo.
  - Cubre todas las tareas identificadas por el equipo para conseguir el objetivo del sprint.
  - Sólo el equipo lo puede modificar durante el sprint.

- El tamaño de cada tarea está en un rango de 4 a 16 horas de trabajo.
- Es visible para todo el equipo. Idealmente en una pizarra o pared en el mismo espacio físico donde trabaja el equipo

- Incremento: Resultado de cada sprint

Reuniones:

- Planificación del sprint: jornada de trabajo previa al inicio del sprint en la que se determina cuál va a ser el trabajo y los objetivos que deben conseguir en la iteración.
- Monitorización del sprint: Reunión diaria breve, de no más de 15 minutos en la que todos los miembros del equipo describen:
  1. El trabajo que realizó el día anterior
  2. El que tiene previsto realizar
  3. Cosas que puede necesitar o impedimentos
- Revisión del sprint: reunión realizada al final del sprint en la que, con una duración máxima de 4 horas el equipo presenta al propietario del producto, clientes, usuarios, gestores… el incremento construido en el sprint
- Demo: el equipo muestra al product owner la funcionalidad terminada (el trabajo realizado durante el sprint). Otros stakeholders están invitados a participar y dar feedback.
- Retrospectiva: el equipo se reúne después de cada iteración para evaluar y mejorar sus métodos (lo bueno, lo malo, lo que hay que mejorar)

Reportes:

- Product Burndown chart: (gráfico de trabajo pendiente) indica que tan rápido el equipo está terminando los requerimientos del product backlog en cada sprint. Monitorea el ritmo de trabajo
- Story o Task burndown chart: indica que tan rápido el equipo está terminando stories o bajando horas en un sprinet.
- Gráfico burn-up: herramienta de gestión y seguimiento para el propietario del producto. Presenta un vistazo de las versiones de producto previstas, las funcionalidades de cada una, velocidad estimada, fechas probables de cada versión, margen de error previsto en las estimaciones y avance real.

***Valores*:**

- Autonomía del equipo
- Respeto en el equipo
- Responsabilidad y auto-disciplina
- Foco en la tarea
- Información transparencia y visibilidad

5) User stories: formato y modelo INVEST

Formato: As as [user role]I want to [goal] so that [benefit]

Ej.Como comprador, quiero agregar libros al carrito de compras para elegir los libros que voy a comprar

- Especifican qué hay que hacer, evitando especificar cómo hacerlo
- Pueden agregarse detalles (breves)
- Pueden tener attachments que definan el contexto
- Conviene que incluyan los criterios de aceptación
- Pueden estar en una herramienta (issue tracker), en un excel, en un repositorio, en google docs o en tarjetas
- Se completa la info en una conversación
- Escribir buenas stories suele ser lo más difícil al arrancar un proyecto de Scrum
- Beneficio: ayuda a pensar desde la perspectiva del usuario

Modelo INVEST

Una buena historia de usuario también sigue el modelo de INVEST: Independiente, Negociable, Estimable, Pequeña (Small), y Testeable.

- ***Independiente*** - una historia debería ser independiente de otras (lo más posible).
- ***Negociable*** - una historia de usuario es negociable. Demasiado detalles inhiben la comunicación (Equipo-Product Owner-Cliente)
- ***Valiosa*** - cada historia tiene que tener valor para el cliente (para el usuario o para el comprador)
- ***Estimable*** - los desarrolladores necesitan poder estimar una historia de usuario para permitir que se pueda priorizar y planificar la historia (storie).
- ***Pequeña*** - una buena historia debe ser pequeña en esfuerzo, generalmente representando no más de 2-3 personas/semana de trabajo.
- ***Testeable*** - una historia necesita poder probarse para que ocurra la etapa de "Confirmación".

6) Métricas ágiles

Velocidad, trabajo y tiempo son las tres magnitudes que mide la gestión de proyectos ágil. Las tres componen la fórmula de la velocidad, definiéndola como cantidad de trabajo realizada en por unidad de tiempo. **Velocidad = Trabajo / Tiempo**

- **Tiempo**
  - Tiempo real: tiempo efectivo de trabajo. *Ej. Para un equipo de cuatro personas con jornada laboral de ocho horas el tiempo real en una semana (cinco días laborables) es: 4 * 8 * 5 = 160 horas*
  - Tiempo ideal: tiempo de trabajo sin ninguna interrupción, pausa, distracción o atención a tareas ajenas a la tarea del sprint que se tiene asignada.

- **Trabajo**
  - Trabajo ya realizado: para medirlo basta con contabilizar las unidades que se empleen: líneas de código, horas trabajadas (reales o ideales). Medir el trabajo realizado NO es una métrica Ágil. LA GESTIÓN ÁGIL DETERMINA EL TRABAJO PENDIENTE DE REALIZAR.
  - Trabajo pendiente de realizar. Scrum mide el trabajo pendiente para:
    - Estimar el esfuerzo y la duración prevista para cada tarea.
    - Determinar el avance del proyecto, y en especial de cada sprint.

  La estrategia empleada por la gestión ágil es:

    - No empeñarse en estimaciones precisas.
    - Estimar con la técnica "juicio de expertos"
    - Descomponer las tareas de los sprints en sub-tareas más pequeñas, si las estimaciones superan los rangos de las 16-20 horas de trabajo.

  - Unidades de trabajo

    Las unidades para medir el trabajo pueden estar directamente relacionadas con el producto, como los tradicionales puntos de función de COCOMO, o a través del tiempo necesario para realizarlo.

    La gestión ágil suele llamar a las unidades que emplea para medir el trabajo "puntos", "puntos de funcionalidad" "puntos de historia"… pero se trata siempre de medición a través del tiempo, no del producto.

- ***Velocidad***

  Velocidad es la magnitud que viene determinada por la cantidad de trabajo realizada en un periodo de tiempo (Timebox) Los equipos que miden el trabajo con tiempo ideal, hablan de "velocidad".

  Decir, por ejemplo, que la velocidad del equipo en el sprint ha sido de 23, se refiere a que han completado tareas que medían en toral 23 story points.

  Si en el sprint siguiente consiguen una velocidad mayor, querrá decir que han logrado programar más story points en el mismo tiempo, o lo que es lo mismo que han conseguido aumentar el porcentaje de tiempo ideal en la jornada de trabajo: han reducido los tiempos de interrupciones, distracciones o dedicados a otras tareas.

7) TDD & ATDD: su objetivo y su comparativa con pruebas unitarias y de aceptación

- **TDD & ATDD** comparten el punto de vista del proceso de desarrollo, pero tienen diferentes herramientas de trabajo:
  - pruebas unitarias en TDD
  - pruebas de aceptación en ATDD.

- **TDD**
  - Cada requisito se representa como un artefacto (user story) al que se le asocian sus correspondientes pruebas unitarias.
    1. Se define el repositorio de historias de usuario, *Product backlog*, que representa los requisitos del sistema.
    2. Los desarrolladores escriben las pruebas unitarias necesarias para satisfacer cada una de las historias de usuario
    3. Implementan el código fuente necesario para superarlas.
  - Las pruebas unitarias no son TDD
  - Pruebas unitarias: su propósito es probar una parte del código aislada del resto. Son una herramienta para representar las pruebas
  - TDD representa el proceso de desarrollo y tiene como objetivo comprobar que la aplicación que va a diseñar funcionará correctamente.

- **ATDD**

  - Toma como punto de referencia al usuario.
    1. Cada historia de usuario tiene asociado un conjunto de criterios de aceptación.
    2. Para cada criterio de aceptación se definen las pruebas de aceptación que se deben superar.
    3. El equipo de desarrollo comienza a escribir el código fuente que es necesario para superar los criterios de aceptación.
    4. También será necesaria la definición de pruebas unitarias, asociadas al código que se implementa que garanticen que el código cumple con los requisitos.
  - la lista completa de historias de usuario (product backlog) junto con sus criterios de aceptación y las pruebas de aceptación desempeñan un papel equivalente a las especificaciones de requisitos software en las metodologías convencionales.
  - Las pruebas de aceptación no son ATDD.
    - Las pruebas de aceptación tienen como objetivo probar la funcionalidad del sistema mientras
    - ATDD representa el proceso de desarrollo y tiene como objetivo que el sistema se construya de acuerdo a la funcionalidad determinada por el usuario. ATDD se lleva a cabo en colaboración con el usuario y en un lenguaje que usuario pueda entender (FIT o Easyaccept).

Unidad 5.1: Métricas para la calidad del software y su integración al proceso de ingeniería de software

1) Definiciones: medición, medida, indicador y métricas.

Los términos medida y medición se pueden utilizar como un nombre o como un verbo, las definiciones de estos términos se puede confundir.

- Medición: Es una medida cuantitativa que permite tener una visión profunda de la eficacia del proceso del sw y de los proyectos utilizando un proceso como marco de trabajo. Se reúnen los datos básicos de calidad y productividad
- Medida: Indicación cuantitativa de la extensión, cantidad, dimensiones, capacidad o tamaño de algunos atributos de algunos atributos de un proceso o producto. La medición es el acto de determinar una medida.
- Indicador: Es una métrica o una combinación de métrica que proporciona una visión profunda del proceso proyecto o producto de sw.
- Métricas: Medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado

2) ¿Cuáles son las cuatro razones para medir los procesos de sw, los productos y los recursos?

Caracterizar:

- Comprender mejor los procesos, productos y recursos, y entornos
- Establecer líneas bases

Evaluar:

- Determinar el estado con respecto al diseño
- Valorar la consecución de los objeticos de calidad
- Evaluar el impacto de la tecnología
- Evaluar las mejoras del proceso en los productos

Predecir:

- Para planificar
- Establecer objetivos alcanzables para el coste, planificación y calidad
- Extrapolación de tendencias (proyecciones y estimaciones basadas en datos históricos)

Mejorar

- Recolección de información cuantitativa
- Problemas de raíz
- Ineficiencias
- Calidad del producto
- Rendimiento del proceso

3) ¿Cuál es el objetivo de todo el proceso de medición?

Contar con un mecanismo objetivo para evaluar productos, procesos y proyectos expresando los resultados en números y colaborando con la mejora continua.

4) ¿Cómo se puede medir la efectividad de un proceso de sw?

Se mide indirectamente. Se extrae un juego de métricas según los resultados que provienen del proceso. Entre los resultados se incluyen medidas de errores detectados e informado a los usuarios finales, producto de

trabajos entregados, esfuerzo humano y tiempo consumido, ajuste con la planificación y otras medidas. Las métricas del proceso también se extraen midiendo las características de tareas específicas de la ingeniería del sw.

5) ¿Que tendría en cuenta al instituir un programa de métrica de proceso?

Las métricas de procesos de sw pueden proporcionar beneficios significativos. Sin embargo si estas se usan mal pueden originar más problemas de lo que solucionan.

Al instituir un programa de métricas de procesos hay que tener en cuenta:

- Usar el sentido común y la sensibilidad organizativa para interpretar datos de las métricas
- Retroalimentar regularmente stakeholders que hayan trabajado en la recopilación de medidas y métricas
- No usar métricas para evaluar a particulares
- Trabajar con profesionales y equipos para establecer objetivos claros y métricas que se vayan a utilizar para alcanzarlos
- No utilizar métricas que amenacen a particulares o equipos
- Los datos de las métricas son un indicador de mejoras de proceso

Unidad 5.2: Mediciones del Software

1) ¿Cuáles son las actividades de un proceso de medición?

- Formulación: Obtener medidas y métricas del sw apropiadas
- Colección: Acumulación de datos necesarios para obtener las métricas formuladas
- Análisis: Calcular métricas mediante el uso de herramientas matemáticas
- Interpretación: Evaluación de los resultados de las métricas para conseguir una visión interna de la calidad
- Realimentación: Recomendación de la interpretación de métrica

2) ¿Cómo podemos valorar la calidad de una métrica de software?

La métrica obtenida y las medidas que conducen a ello deberían ser:

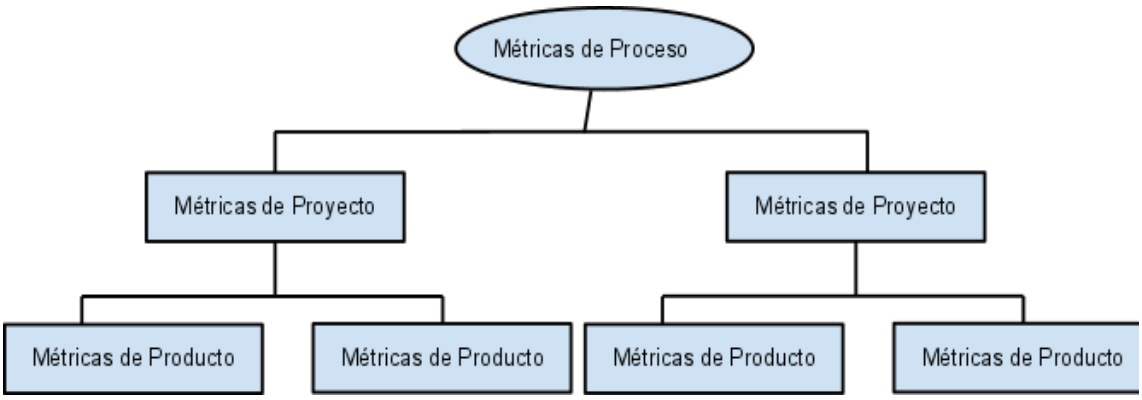
- Simples y fácil de calcular
  - Facilidad para obtener una métricas
  - Cálculo con poco esfuerzo o tiempo
- Empírica e intuitivamente persuasivas
- Consistentes y objetivas
  - Resultados no ambiguos
- Consistentes en el empleo de unidades y tamaños
- Independientes del lenguaje de programación
- Un mecanismo eficaz para la realimentación de calidad

- Conducir a un producto final de mayor calidad

3) ¿Cuál es la diferencia entre medidas directas e indirectas?

- ***Medidas directas***
  - Entre las medidas directas del proceso de la ingeniería del software se incluyen el coste y el esfuerzo aplicados.
  - Entre las medidas directas del producto se incluyen las líneas de código (LDC) producidas, velocidad de ejecución, tamaño de memoria, y los defectos informados durante un período de tiempo establecido.
- Entre las ***medidas indirectas*** se incluyen la funcionalidad, calidad, complejidad, eficiencia, fiabilidad, facilidad de mantenimiento y muchas otras capacidades.

4) ¿Cuál es la relación entre las métricas de proceso, proyecto y producto?



La relación entre las métricas de proyecto, proceso y producto se muestra en la figura. Como se puede observar, el proceso SW constituye la base a partir de la cual se realiza el trabajo de una organización. Dichos procesos se aplican en la práctica en forma de proyectos. Como resultado de la ejecución de proyectos concretos se utilizan recursos y se obtienen productos. Por lo tanto, para establecer un marco de medición dentro de una organización, es necesario definir, recoger y analizar métricas sobre el proceso, el proyecto y recursos asociados así como el producto SW.

5) ¿Qué implican las mediciones de un proceso? ¿Cuál es el objetivo de medición de un proyecto?

Las mediciones del proceso implican las mediciones de las actividades relacionadas con el SW, siendo algunos de sus atributos típicos el esfuerzo, el coste y los defectos encontrados.

Cuando se mide el proyecto, el objetivo fundamental que se pretende es el de reducir el coste total y el tiempo de desarrollo del mismo

6) ¿Qué le permiten los indicadores a un líder de proyecto? ¿Cuáles son los métodos prácticos genéricos para un líder de proyecto?

Los indicadores de proyecto permiten al administrador de SW:

- Evaluar el estado del proyecto en curso
- Realizar un seguimiento de los riesgos potenciales
- Detectar las areas de problemas antes que se conviertan en “criticas”
- Ajustar el flujo y las tareas de trabajo
- Evaluar la habilidad del equipo de proyecto en controlar la calidad de los productos de trabajo de la Ing SW

Los métodos prácticos genéricos para gerentes de proyectos de software, entre clientes de SPR, asemejan los siguientes:

- un gerente de proyecto por cada ocho miembros del personal técnico
- Un gerente de proyecto de tiempo completo, por cada 1500 puntos función.
- Un gerente de proyecto por aproximadamente cada 150000 instrucciones de código fuente
- La administración de un proyecto comienza antes de los requisitos y se extiende hasta después de terminar el proyecto
- Trabajo de administración del proyecto = 35% de tiempo de administración entregable
- Trabajo del personal = 30% de tiempo de administración entregable
- Reuniones con otros gerentes o clientes = 22% de tiempo de administración entregable
- Trabajo departamental = 8% de tiempo de administración entregable
- Trabajo misceláneo: 5% de tiempo de administración entregable

7) ¿En qué se centra la medición del producto? ¿Cuáles son sus salidas?

La medición del producto está centrada en evaluar la calidad de los entregables. Los productos del sw son las salidas del proceso de producción del sw, que incluyen, todos los artefactos entregados o documentos que son productos durante el ciclo de vida del sw.

8) ¿Cuáles son las tres ventajas de adquirir y usar herramientas de estimación de costos?
Las tres ventajas de adquirir y usar herramientas comerciales de estimación de costos de software son triples:

- La curva de aprendizaje para manejar las herramientas de manera efectiva no es trivial
- Las herramientas necesitan calibrarse para ajustarlas a condiciones locales
- Algunas de las mejores herramientas de estimación de costos son bastantes costosas

9) ¿Qué métricas aplicarías en la calidad de las especificaciones?

- Especificidad (ausencia de ambigüedad)
- Compleción
- Corrección
- Comprensión
- Capacidad de verificación
- Consistencia interna y externa
- Capacidad de logro.
- Concisión.
- Trazabilidad

- Capacidad de modificación.
- Exactitud.
- Capacidad de reutilización.

Además, las especificaciones de alta calidad deben estar almacenadas electrónicamente, ser ejecutables o al menos interpretables, anotadas por importancia y estabilidad relativas, con su versión correspondiente, organizadas, con referencias cruzadas y especificadas al nivel correcto de detalle.

Unidad 5.3: Introducción a la estimación de costos de software

1) ¿Cómo funcionan las herramientas de estimación de costos de software?

Las herramientas de estimación de costos de software funcionan tomando como base los siguientes puntos:

1. Contienen bases de conocimiento de cientos o miles de proyectos de software.
2. Realizan predicciones de tamaño, que las herramientas genéricas son incapaces de realizar.
3. Ajustan automáticamente estimaciones basadas en herramientas, lenguajes y procesos.
4. Predicen calidad y confiabilidad.
5. Predicen costos de mantenimiento y soporte despuès de implementadas.
6. Predicen y preveen problemas, mucho tiempo antes de que estos ocurran en realidad.

Lo que se realiza primordialmente es determinar el tamaño del proyecto basándose en las bases de conocimiento (principalmente en aplicaciones comerciales) o en entradas que se le dan a la herramienta por parte del gerente del proyecto o encargado de realizar estos cálculos. Luego se ingresan otros atributos que pueden afectar a estas estimaciones.

2) ¿Cuáles son los principios básicos de las herramientas comerciales de estimación de costos de software? (Fórmula)

Las herramientas comerciales se basan en enormes bases de conocimientos de proyectos anteriores, por lo que permiten heredar los atributos que afectan a las estimaciones de proyectos anteriores u otros proyectos similares. La figura que sigue ilustra los principios básicos de las herramientas comerciales modernas de estimación de costos de software:

3) ¿Cuáles son los atributos que pueden afectar el resultado de una estimación?

Luego de determinado el tamaño se ingresan otros atributos que pueden afectar los resultados de la estimación, estos suelen ser:

1. Velocidad a que pueden cambiar los requisitos del proyecto.
2. Experiencia del equipo de desarrollo con este tipo de proyecto.
3. Proceso o métodos empleados para desarrollar el proyecto.
4. Actividades específicas a realizarse durante el desarrollo.
5. Número de incrementos, iteraciones o "carreras" que se utilizarán.
6. El o los lenguajes de programación a ser manejados.
7. Presencia o ausencia de artefactos reutilizables.
8. Suites de herramientas de desarrollo que se usarán para desarrollar el proyecto.
9. Entorno o ergonomía del espacio de trabajo en la oficina.
10. Separación geográfica del equipo en múltiples lugares.
11. Presión de los tiempos límites ejercida en el equipo por clientes o ejecutivos.
12. Obligaciones contractuales en términos de costos, fechas, defectos o características.

4) ¿Qué actividades se incluye al estimar?

Las actividades que se incluyen al estimar son:

- Las estimaciones de costos precisas deben iniciar con predicciones de calidad, pues los costos de eliminación de defectos suelen ser más elevados que cualquier otro.

- En segundo lugar, como elementos de costo importantes, se encuentran gastos y esfuerzos dedicados a la producción de documentos en papel: planes, especificaciones, manuales de usuario, etc.

- Muchos proyectos grandes sitúan costos y fechas límite en el manejo de "requisitos progresivos" o nuevas características agregadas al proyecto, tras la fase de requisitos. Todos los desarrollos de software crecerán por requisitos progresivos y, por tanto, este factor debe ser parte integral de las estimaciones de todos los proyectos importantes.

- Una omisión frecuente en las estimaciones de costos es la exclusión accidental de viáticos (líneas aéreas, hoteles, etc.), para reuniones entre equipos de desarrollo en diferentes ciudades y países. Los viáticos pueden exceder el costo de la redacción del código de forma significativa y el tema no puede omitirse de manera accidental.

- Muchas estimaciones de costos de software, cubren sólo las actividades centrales del desarrollo de software y omiten temas tales como administración y soporte a proyectos (secretarias, administración, etc.). Estas actividades auxiliares son parte del proyecto y pueden ascender, en algunos casos, a 20% de los costos totales.

- Es muy común omitir las contribuciones de especialistas si sus habilidades sólo se necesitan sólo en una etapa del ciclo de desarrollo del software. Algunos grupos de especialistas que tienden a ser omitidos accidentalmente de las estimaciones de los costos de software, integrarían las áreas de control de

calidad, redacción técnica, puntos de función, administración de bases de datos, optimización de rendimiento, redes y administración de sistemas. Las aportaciones combinadas de estos y otros especialistas pueden totalizar más de 20% del costo total de para el desarrollo de software y no deben omitirse de forma accidental.

- La omisión más común de las estimaciones internas de los costos de software para sistemas de información, son costos en que incurren los usuarios durante la definición de requisitos, creación de requisitos, revisiones de estado, revisiones de fases, documentación, inspecciones, pruebas de aceptación y otras actividades en que los desarrolladores tienen un rol clave. Como los representantes de usuarios no suelen considerarse parte del equipo del proyecto, sus contribuciones al proyecto rara vez se incluyen en las estimaciones del costo del software y estudios de medición. El esfuerzo real que aportan los usuarios a proyectos de desarrollo de software importantes puede acercarse al 20% del trabajo total en algunos casos.

- En el caso de muchos proyectos, el mantenimiento después de la entrega cuesta más que el desarrollo de la aplicación misma. No es recomendable detener la estimación en el punto de entrega del software, sin incluir cuando menos 5 años de estimaciones de mantenimiento y mejoras.

5) ¿Qué tipo de actividad es la estimación de costos?¿De dónde deriva?

La estimación de costos de software no es una actividad "autónoma". Las estimaciones se derivan en gran medida de requisitos del proyecto y otros atributos asociados con el proyecto. Una estimación de costos es precursora de presupuestos departamentales y sirve también como documento base para comparar costos acumulados contra proyectados

6) ¿Qué estimaciones se realizan durante el proceso de desarrollo?

Para cualquier proyecto más grande que uno trivial, se elaborarán múltiples estimaciones de costos durante el proceso de desarrollo, incluyendo, pero limitado, a las siguientes:

- Una estimación aproximada de requisitos del proyecto.
- Una estimación formal inicial derivada de requisitos de proyecto.
- Una o más estimaciones a la mitad del ciclo de vida, reflejando cambios en los requisitos.
- Una acumulación final de costos, empleando datos históricos del proyecto.

7) ¿Cuáles son los consejos para realizar las estimaciones?

- Sea preciso.
- Sea conservador.
- Base la estimación en datos históricos sólidos.
- Procure calidad, pues las características del software afectarán fechas límite y costos.
- Procure no incluir documentos en papel, ya que pueden valer más que el código fuente.
- Incluya administración de proyectos.
- Incluya los efectos de requisitos progresivos.
- No exagere el efecto de las herramientas, lenguajes y métodos.
- Lleve las fases anteriores a estimaciones de costos, a nivel de actividades.

- Esté preparado para defender las suposiciones de su estimación.

8) ¿Cuál es el objetivo de GQM?

El objetivo de estos marcos de trabajo es proporcionar las referencias necesarias para poder llevar a cabo el proceso de medición de una forma efectiva y sistemática, en base a una serie de objetivos.

Unidad 5.4: Estimación de costos de software

1) ¿Cuales son las principales tareas que debe realizar una herramienta de métricas?

- Adquisición de datos: conjunto de métodos y técnicas que permiten la obtención de datos necesarios para realizar la medición. Esta tarea presenta las siguientes posibilidades:
  - Manual
  - Semiautomática
  - Automática
  - Programable
- Análisis de las mediciones: incluye la habilidad para almacenar, recuperar, manipular y llevar a cabo el análisis de los datos. Esta tarea supone realizar las siguientes actividades:
  - Almacenamiento de los datos de la medición.
  - Recuperación de los datos de la medición.
  - Análisis Aritmético de resultados.
  - Análisis estadístico de resultados.
- Presentación de los datos: formatos que facilita la herramienta de medición para generar la documentación obtenida. Destacan las siguientes posibilidades de representación:
  - Tablas.
  - Gráficos.
  - Exportación de archivos a otras aplicaciones.

2) ¿Cómo se clasifican las herramientas?

- Herramientas universales de métricas: aquellas que están diseñadas para soportar el trabajo con métricas y enfocadas a su obtención, análisis y presentación. Además estas herramientas deben proveer al usuario de:
  - Una interfaz de usuario flexible que permita realizar una obtención de datos exacta y efectiva.
  - Una amplia variedad de algoritmos o estándares para el análisis de un gran conjunto de métricas
  - La capacidad de generar un conjunto de informes flexibles y orientado a la representación grafica que además permita personalizar su aspecto en función de los requisitos del usuario.
- Herramientas especializadas de métricas: aquellas que están diseñadas para soportar:
  - Otro tipo de actividad o funciones aparte de la medición, pero que aportan ciertas métricas especificas acerca del ciclo de vida del software, por ejemplo las herramientas de gestión del proyecto.

- Una métrica especifica o conjunto concreto de métricas, por ejemplo las herramientas de medición de la complejidad del código

3) ¿Cuales son los tipos de herramientas especializadas?

- Herramientas de estimación software: aquellas herramientas que permiten estimar esfuerzo y duración del proyecto en función de parámetros de tamaño y del entorno de desarrollo software

- Herramientas de gestión de proyecto: aquellas herramientas que soportan la planificación, seguimiento y gestión de un proyecto basándose en los recursos (tiempo, personal, etc.)

- Herramientas de contabilidad temporal: aquellas herramientas que aportan información temporal sobre la ejecución carga del código

- Herramientas de análisis de código fuente: aquellas herramientas que recorren el código fuente escrito en un determinado lenguaje de programación y cuentan líneas de código, puntos función, etc. y calculan ciertas métricas de complejidad.

- Herramientas de seguimiento de cambios y defectos: las herramientas de seguimientos de cambios permiten almacenar y analizar los cambios realizados en código como resultado de la corrección de defectos o actividades de mejora. Las herramientas de seguimiento de defectos permiten detectar los defectos que presenta el código aportando información sobre la severidad del defecto, el tipo de defecto o la localización dentro del código.