Arquitecturas de software

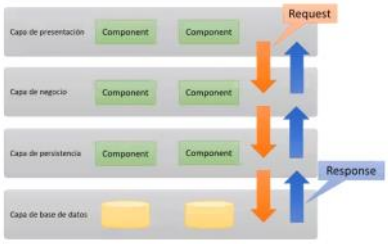
Tuesday, April 18, 2023 7:26 PM

Arquitectura de Software

- Según el Software Engineering Institute (SEI), la arquitectura de software hace referencia a "las estructuras de un sistema, compuestas de elementos con propiedades visibles de forma externa y las relaciones que existen entre ellos".
- La arquitectura de software se refiere a las partes que vamos a construir y la forma en la que las vamos a combinar y juntar para poder trabajar con ellas.
- Veamos los tipos y los patrones de arquitectura de SW más conocidos.

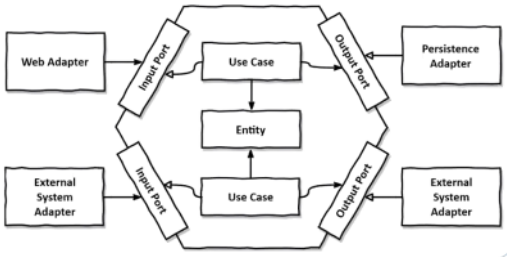
Arquitectura en capas (tipo)

- La arquitectura por capas nace para suplir los problemas derivados de la arquitectura spaghetti (que no es una arquitectura realmente) y su objetivo es contar con capas que tienen diferentes usos.
- Por ejemplo, mientras que una de ellas se encarga de la visualización de datos, otra tiene que servir para acceder a la base de datos. De esta manera, cada tarea se reparte en una capa diferente, de modo que se facilita el trabajo en ellas.



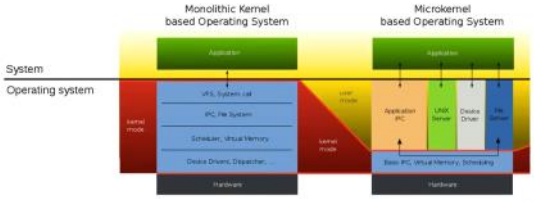
Arquitectura hexagonal (tipo)

- La arquitectura hexagonal aísla las entradas y salidas de aplicación de la lógica interna de la aplicación.
- Dado el aislamiento, se generan partes independientes que no dependen de los cambios externos, de esta forma, estos pueden cambiarse de forma individual sin afectar al resto.



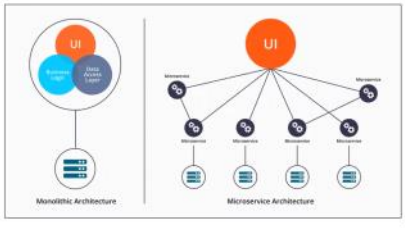
Software microkernel (patrón)

- Se utiliza cuando se realizan sistemas con componentes intercambiables.
- Usualmente se ocupa en la implementación de aplicaciones basadas en productos, es decir, aquellas que están empaquetadas y disponibles para su posterior descarga.
- Permite añadir características adicionales como plug-ins y es muy flexible y extensible.
- Se usa para aplicaciones que precisan datos de diferentes fuentes y para aplicaciones de flujo de trabajo.



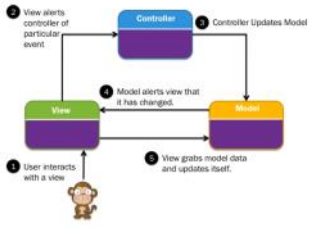
Microservicios (patrón)

- En lo simple, lo que hace este patrón es escribir multitud de solicitudes que van a funcionar juntas.
- Su principal ventaja es la posibilidad de escribir, mantener y desplegar cada microservicio de forma individual.
- Se utiliza, sobre todo, para webapps con pequeños componentes, centros de datos no muy grandes y el desarrollo rápido de aplicaciones web.



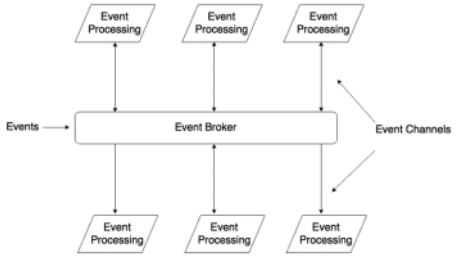
Modelo Vista Controlador - MVC (patrón)

- La arquitectura de software basada en el patrón Modelo-Vista-Controlador (MVC), divide el desarrollo del sistema en conceptos.
- Su objetivo es separar la lógica del negocio de la presentación.
- Pudiera parecer similar al Modelo de N-Capas, pero a diferencia de este la lógica puede estar presente en todos los componentes y cada componente se puede comunicar con los demás indistintamente, sin tener que pasar por alguno de ellos de manera obligatoria.



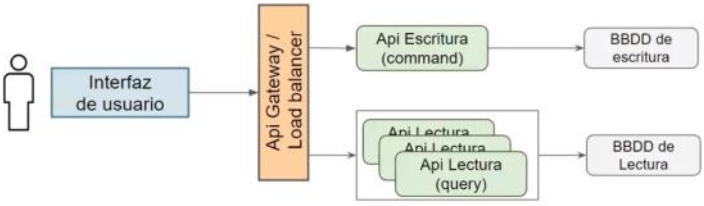
Event-based pattern (patrón)

- Es una arquitectura asíncrona.
- Se utiliza para desarrollar sistemas altamente escalables (es su gran ventaja).
- Se basa en componentes de procesamiento de eventos de un solo propósito.
- Se utiliza, principalmente, para cuestiones como las interfaces de usuario.



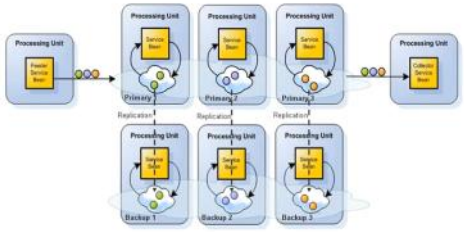
Patrón CQRS (patrón)

- CQRS: Segregación de Responsabilidades de Comandos y Consultas.
- Separa las operaciones de lectura y actualización de un almacén de datos.
- Se utiliza para maximizar el rendimiento, la escalabilidad y la seguridad de una aplicación.
- Permite que el sistema evolucione mejor con el tiempo y no se vea dañado por los comandos de actualización.



Arquitectura basada en espacio (patrón)

- Este patrón se usa para la resolución de problemas de escalabilidad y concurrencia.
- Para ganar escalabilidad se elimina la restricción de la base de datos central sustituyéndola por cuadrículas de datos replicados en memoria.
- Su principal ventaja es que consigue responder de forma rápida a los cambios y se usa para manejar datos de gran volumen (Por ej. Registros de usuarios).



Historias de usuario

Tuesday, April 18, 2023 6:26 PM

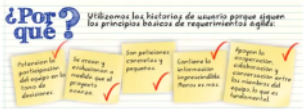
Historias de Usuario

¿Qué es una Historia de Usuario?

"Una historia de usuario es una explicación general e informal de una función de software escrita desde la perspectiva del usuario final. Su propósito es articular cómo proporcionará una función de software valor al cliente".

Por lo general, una historia de usuario seguirá el siguiente formato:

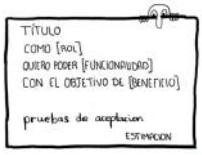
Como [perfil], quiero [objetivo del software], para lograr [resultado]"



Redacción de Historias de Usuario

Si bien es cierto que no existe una "receta de cocina", se sugiere identificar los siguientes componentes:

- Perfil: Se refiere al rol del usuario final
- Necesidad: Se asocia al objetivo que tiene la función del Software para el usuario final
- Propósito: Hace referencia a la propuesta de valor que tendrá el Software para el usuario final



Identificación del Perfil

Para identificar el perfil del usuario final es necesario al público objetivo y tomar en cuenta a quién o quienes impactará la función de software.

Sugerencia de preguntas para identificar perfil del usuario:

- ¿Para quién estamos creando esta función de software?
- ¿Qué tipo de características del producto quiere el usuario final?
- ¿Cuáles son los datos vinculados al entorno del usuario final? (demográficos, poblacionales, etc.).

Importante

Puede haber varios perfiles en una historia de usuario determinada dependiendo del tamaño del público objetivo.

Ejemplo de perfil

Mauricio Hidalgo, un Jefe de Proyecto que tiene a cargo un equipo de desarrollo de cinco personas.

Descripción de la necesidad

Responde a cómo el usuario final utilizará una funcionalidad del software y por qué. Esto es fundamental para que tu equipo entienda por qué el público objetivo elegiría usar tu función.

Sugerencia de preguntas para describir la necesidad:

- ¿Qué está tratando de lograr el usuario final?
- ¿Cómo ayudará esta funcionalidad al usuario final para que pueda lograr sus objetivos?

Importante

Se debe considerar lo que el usuario final está buscando y la manera en que tu software lo ayudará a alcanzar sus objetivos por sobre las características específicas del software.

Ejemplo de necesidad

Ayudar a los desarrolladores a comprender como cumplir con sus entregas parciales contribuye al desarrollo de una solución global.

Definición del propósito

Para definir el propósito es importante analizar el panorama general de operación del software considerando cómo la función de software se ajusta a tus objetivos internos.

Sugerencia de preguntas para definir el propósito:

- Cuál es el beneficio de la función de software?
- ¿Cuál es el problema que se está resolviendo?
- ¿Cómo se puede acoplar esta funcionalidad en metas más amplias?

Importante

El propósito es definir el valor de tu función del software en relación con los objetivos generales.

Ejemplo de propósito

Disminuir los tiempos de desarrollo a través del control de Sprints.

Ejemplo de Historias de Usuario

Ejemplo 1

Como usuario de una cuenta de correo electrónico, espero que mis datos se puedan guardar en las configuraciones de mi equipo personal para lograr un acceso más rápido a mi bandeja de entrada.

Ejemplo 2

Como comprador de una tienda online quisiera ver las críticas de un producto en particular para decidir si lo compro.

Ejemplo 3

Como docente de una asignatura quiero registrar las notas para obtener los promedios.

Consideración importante: INVEST

INVEST es un acrónimo asociado a criterios para ayudar a la redacción de Historias de Usuario.

- Independiente: Una Historia de usuario no debe depender de otras tareas (en lo ideal).
- Negociable: Debe dejar espacio para la discusión.
- Valiosa: Debe transmitir valor a través de ayudar a objetivos de largo plazo.
- Estimable: Se debe poder aproximar su ajuste a un sprint.
- Pequeña (small): El trabajo se debe poder realizar en poco tiempo.
- Comprobable (testable): Debe cumplir criterios de aceptación y pasar pruebas para verificar su calidad.

Requisitos funcionales

Monday, April 17, 2023 5:28 PM

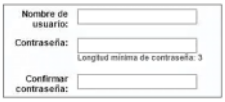
Requerimientos Funcionales

Definiciones posibles

- Se refiere a aquellos requisitos que definen una función del software o de sus componentes.
- Son aquellos que establecen los comportamientos del Software ante input determinados.

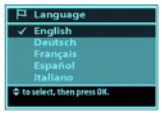
¿Qué establecen los Requerimientos Funcionales?

- Servicios o funciones que proveerá el sistema
- Interacción entre el sistema y su entorno



Un requerimiento funcional típico contiene:

- Identificador y/o Nombre
- Resumen y Descripción
- Prioridad



Ejemplo de Requerimientos Funcionales

Número	Requerimiento	Descripción	Prioridad
RF1	LA ADMINISTRACION LLEVARA UN CONTROL DE NOTAS DE LOS ALUMNOS	El sistema tendrá las notas de los alumnos en cada curso que ha llevado y que esta cursando actualmente,	5
RF2	EL ADMINISTRADOR PODRA MODIFICAR LAS NOTAS	El sistema deberá permitir la modificación de las notas del curso del ciclo vigente	5
RF3	VIZUALIZAR LA RELACION DE CURSOS Y NOTAS.	El sistema proporcionara al Profesor y al Alumno la información de notas de los cursos del ciclo vigente	4
RF4	TENER UN CONTROL DE LA INFORMACION DE LOS PAGOS	El sistema proporcionara los datos de los pagos realizados por el alumno.	4
RF5	GENERACION DE INFORMES Y REPORTES	Con los informes se podrá obtener resultados detallados sobre las notas del curso, notas por evaluación, además del promedio final, grado académico, clasificarlos por alumno, área, fecha, etc. Estos podrán ser impresos.	4
RF6	GENERACION DE USUARIOS CURSOS Y CICLOS PARA LA ADMINISTRACIOND EL SISTEMA	El sistema permitirá el ingreso de nuevos usuarios en cualquiera de las tres categorías, ciclos academicos y cursos para cada programa.	4

<cal

Requisitos no funcionales

Requerimientos no Funcionales

Definiciones posibles

- Se refiere a criterios que se pueden utilizar para evaluar la operación de un sistema en lugar de sus comportamientos específicos
- Son una declaración de propiedades emergentes del sistema. Por ejemplo: Disponibilidad, Fiabilidad, Tiempos de respuesta, entre otras.

NOTA AL MARGEN

También son conocidos como "atributos de calidad" en el ámbito de la Arquitectura de Software.

Discutamos con un ejemplo

Supongamos que tenemos mucho presupuesto y vamos a construir una casa: ¿Qué preguntas podríamos hacer a nuestro cliente/usuario?

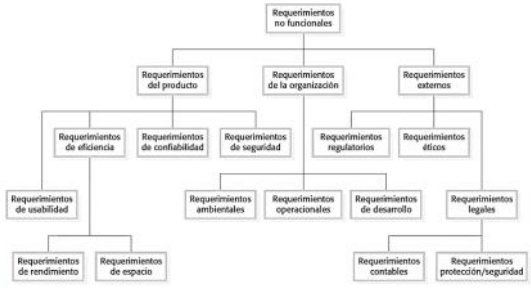
Ejemplo: Requerimientos no Funcionales

Número	Requerimiento	Descripción	Prioridad
RNF1	USABILIDAD	Debe ser fácil de usar. Con ayudas e interfaces intuitivas.	5
RNF2	SEGURIDAD	El ingreso al sistema estará restringido bajo contraseñas cifradas y usuarios definidos.	5
RNF3	PORTABILIDAD	El sistema debe brindar comodidad al usuario y a otras áreas que trabajan o necesitan del Área de personal. Por ejemplo El Sistema de Pago y Planillas no debe tener problemas en acceder al Sistema de Personal.	5
RNF4	MULTIPLATAFORMA	El sistema deberá funcionar en distintos tipos de sistemas operativos y plataformas de hardware.	3
RNF5	RENDIMIENTO	El sistema debe soportar el manejo de gran cantidad de información durante su proceso.	3
RNF6	DESEMPEÑO	El sistema no presentara problemas para su manejo e implementación.	1

Ejemplo obtenido de https://es.slideshare.net/unimauro/sistema-de-gestion-de-notas

Tipos de Requerimientos no Funcionales

Según Sommerville, tenemos los siguientes tipos de RNF



Tipos de Requerimientos no Funcionales

SQuaRE: System and Software Quality Requirements and Evaluation

Es la descripción de como los modelos de calidad pueden ser usados para los requerimientos de calidad de software.

ISO/IEC 25010

La información de estas diapositivas fue adaptada de iso25000.com para fines exclusivamente académicos

ISO/IEC 25010 - Adecuación Funcional

Capacidad del producto para proporcionar funciones que satisfacen las necesidades declaradas e implícitas, cuando el producto se usa en las condiciones especificadas.

- ***Completitud funcional***

Grado en el cual el conjunto de funcionalidades cubre todas las tareas y los objetivos del usuario especificados.

- ***Corrección funcional***

Capacidad para proveer resultados correctos con el nivel de precisión requerido.

- ***Pertinencia funcional***

Capacidad para entregar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados.

ISO/IEC 25010 - Eficiencia de desempeño

Representa el desempeño relativo a la cantidad de recursos utilizados bajo determinadas condiciones.

- ***Comportamiento temporal***

Los tiempos de respuesta, procesamiento y ratios de throughput de un sistema cuando lleva a cabo sus funciones bajo condiciones determinadas en relación con un banco de pruebas (benchmark) establecido.

- ***Utilización de recursos***

Las cantidades y tipos de recursos utilizados cuando el software lleva a cabo su función bajo condiciones determinadas.

- ***Capacidad***

Grado en que los límites máximos de un parámetro de un producto o sistema software cumplen con los requisitos.

ISO/IEC 25010 – Compatibilidad

Capacidad de dos o más sistemas o componentes para intercambiar información y/o llevar a cabo sus funciones requeridas cuando comparten el mismo entorno hardware o software.

- ***Coexistencia***

Capacidad del producto para coexistir con otro software independiente, en un entorno común, compartiendo recursos comunes sin detrimento.

- ***Interoperabilidad***

Capacidad de dos o más sistemas o componentes para intercambiar información y utilizar la información intercambiada.

ISO/IEC 25010 – Usabilidad

Capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario, cuando se usa bajo determinadas condiciones.

- ***Capacidad para reconocer su adecuación***

Capacidad del producto que permite al usuario entender si el software es adecuado para sus necesidades.

- ***Capacidad de aprendizaje***

Capacidad del producto que permite al usuario aprender su aplicación.

ISO/IEC 25010 – Fiabilidad

Capacidad de un sistema o componente para desempeñar las funciones especificadas, cuando se usa bajo unas condiciones y periodo de tiempo determinados.

- ***Disponibilidad***

Capacidad del sistema o componente de estar operativo y accesible para su uso cuando se requiere.

- ***Capacidad de recuperación***

Capacidad del producto software para recuperar los datos directamente afectados y reestablecer el estado deseado del sistema en caso de interrupción o fallo.

- ***Tolerancia a fallos***

Capacidad del sistema o componente para operar según lo previsto en presencia de fallos hardware o software.

- ***Madurez***

Capacidad del sistema para satisfacer las necesidades de fiabilidad en condiciones normales.

ISO/IEC 25010 – Seguridad

Capacidad de protección de la información y los datos de manera que personas o sistemas no autorizados no los puedan leer y/o modificar.

- ***Confidencialidad***

Capacidad de protección contra el acceso de datos e información no autorizados, ya sea accidental o deliberadamente.

- ***Autenticidad***

Capacidad de demostrar la identidad de un sujeto o un recurso.

- ***Integridad***

Capacidad del sistema o componente para prevenir accesos o modificaciones no autorizados a datos o programas de ordenador.

ISO/IEC 25010 – Mantenibilidad

Esta característica representa la capacidad del producto software para ser modificado efectiva y eficientemente, debido a necesidades evolutivas, correctivas o perfectivas.

- ***Modularidad***

Capacidad de un sistema o programa de ordenador (compuesto de componentes discretos) que permite que un cambio en un componente tenga un impacto mínimo en los demás.

- ***Reusabilidad***

Capacidad de un activo que permite que sea utilizado en más de un sistema software o en la construcción de otros activos.

- ***Capacidad para ser probado***

Facilidad con la que se pueden establecer criterios de prueba para un sistema o componente y con la que se pueden llevar a cabo para determinar si se cumplen.

ISO/IEC 25010 – Portabilidad

Capacidad del producto o componente de ser transferido de forma efectiva y eficiente de un entorno hardware, software, operacional o de utilización a otro.

- ***Adaptabilidad***

Capacidad del producto que le permite ser adaptado de forma efectiva y eficiente a diferentes entornos determinados de hardware, software, operacionales o de uso.

- ***Capacidad para ser instalado***

Facilidad con la que el producto se puede instalar y/o desinstalar de forma exitosa en un determinado entorno.

- ***Capacidad para ser reemplazado***

Capacidad del producto para ser utilizado en lugar de otro producto software determinado con el mismo propósito y en el mismo entorno.

Dominio de Aplicación

Un dominio de aplicación constituye un límite de aislamiento para la seguridad, el control de versiones, la confiabilidad y la codificación.

En simple, son las actuales categorías de SW que se pueden desarrollar:

- Software de sistemas (compiladores, editores, etc) → Alta interacción con el HW
- Software de aplicación (puntos de venta, exploradores) → Hacen funciones específicas
- Software de ingeniería y ciencias (Matlab, Stellarium) → Uso diverso en investigación
- Software incrustado (termostatos, sensores) → Ejecutan funciones limitadas
- Software de línea de productos (Netflix, PS Store, Excel) → Capacidad específica para muchos consumidores diferentes
- Aplicaciones web (SAP Business One, Mercadolibre) → Webapps que grupan muchas aplicaciones
- Software de inteligencia artificial (Tensorflow y Keras) → Requieren gran capacidad de HW

Requerimientos de Software

Un requerimiento es la definición de las funciones, capacidades o atributos intrínsecos de un sistema de software.

Características de los Requerimientos

Característica	Descripción
Completo.	Todos los ítems necesarios para la especificación de la solución están incluidos.
Correcto.	Cada ítem es libre de errores.
Preciso.	no ambigüo y claro. Cada ítem es exacto y no vago; hay una sola interpretación; el significado de cada ítem es entendido; la especificación es fácil de entender.
Consistente.	Ningún ítem entra en conflicto con otro de la especificación.
Relevante.	Cada ítem es pertinente al problema y su solución.
Probable (testable).	Durante el desarrollo del programa y las pruebas de aceptación, es posible determinar si el ítem ha quedado satisfecho.
Factible.	Cada ítem puede ser implementado con las técnicas, herramientas, recursos y personal disponible y dentro de las limitaciones de costo y calendarización.
Registrabilidad (Traceable).	Cada ítem puede ser seguido durante cada etapa.
Libre de detalle de diseño.	La especificación de requerimientos son declaraciones de los requerimientos que se deben satisfacer por la solución del problema y no se deben ocultar por soluciones propuestas al problema.
Manejable.	Los requerimientos son expresados de tal manera que cada ítem puede ser cambiado sin causar un gran impacto a los demás ítems.

Requerimientos de Usuario

Son descripciones de los requerimientos funcionales y no funcionales de tal forma que sean compresibles por los usuarios del sistema sin conocimiento técnico detallado.

Tema:	Organizar Torneo			
Grupo:	Ronald Cárdenas, Israel Rey, Carlos Ojeda			
Id	Descripción	Prioridad	Tipo	Riesgo
RU001	El sistema debe permitir el registro de los equipos de futbol.	Alta	F	ALTA
RU002	El sistema deberá ser capaz de poder planificar los encuentros de los equipos (partidos de futbol).	Media	F	Alta
RU003	Se podrá generar las tablas de posiciones de forma automática.	Alta	F	Alta
RU004	El sistema deberá identificara al mejor goleador	Media	F	Media
RU005	El sistema podrá publicar información acerca del torneo en curso en la web.	Media	F	Alta
RU006	Facilidad de uso: El sistema se diseñara de tal forma que permita al usuario una navegación intuitiva.		NF	
RU007	Seguridad: Las actualizaciones solo podrán ser efectuadas por el organizador del torneo.		NF	
RU008	Portabilidad: Se deberá tener acceso al sistema desde cualquier terminal con internet.		NF	

Requerimientos del usuario → Gerentes del cliente, Usuarios finales del sistema, Ingenieros del cliente, Gerentes de los contratistas, Arquitectos del sistema

****Ejemplo de Israel Rey, QA test Engineer at Universidad Técnica Particular de Loja. extraído para fines académicos*

Requerimientos de Sistema

Son versiones extendidas de los requerimientos del usuario dado que agregan detalle y explican como el sistema debe proporcionar los requerimientos del usuario.

Id	Descripción	Prioridad	Tipo	Riesgo
RU001	El sistema debe permitir el registro de los equipos de futbol.	Alta	F	ALTA
RU002	El sistema deberá ser capaz de poder planificar los encuentros de los equipos (partidos de futbol).	Media	F	Alta

Requerimientos del sistema → Usuarios finales del sistema, Ingenieros del cliente, Arquitectos del sistema, Desarrolladores de software

Deriva: RU001
- Al registrar un equipo de futbol el sistema presentara al usuario un formulario solicitando los siguientes campos :
  - Nombre del equipo
  - Ciudad
  - Nombre del Presidente del equipo
  - Nombre del Director Técnico
- Para el registro de los jugadores el sistema solicitara: el nombre, nacionalidad, fecha de nacimiento del futbolista, número de camiseta y rol que desempeña el jugador en la cancha (defensa, portero, etc.).
- El sistema debe almacenar toda esta información en una base de datos relacional.

****Ejemplo de Israel Rey, QA test Engineer at Universidad Técnica Particular de Loja. extraído para fines académicos*

Requerimientos de Sistema

Importante

- No deben tratar de cómo se debe diseñar o implementar.
- Deben describir el comportamiento externo del sistema y sus restricciones operativas.
- Son requisitos funcionales pues reflejan las operaciones que el sistema llevará a cabo.

Planificar encuentros SRS002.
Tipo funcional
Deriva: RU002
- El sistema deberá contar con formularios que permitan al usuario especificar el número de equipos que tendrá cada grupo o división. El sistema contara con dos opciones para la organización de los encuentros.
  - Planificación de los partidos de forma aleatoria
  - Planificación de los partidos de forma manual.
- Para la creación del calendario de futbol el sistema debe presentar la siguiente información
  - Grupos / divisiones
  - Nombres de los equipos.
  - Fecha y hora del encuentro
  - Estadio
  - Ciudad

Id	Descripción	Prioridad	Tipo	Riesgo
RU001	El sistema debe permitir el registro de los equipos de futbol.	Alta	F	ALTA
RU002	El sistema deberá ser capaz de poder planificar los encuentros de los equipos (partidos de futbol).	Media	F	Alta

****Ejemplo de Israel Rey, QA test Engineer at Universidad Técnica Particular de Loja. extraído para fines académicos*

Requerimientos de Dominio

Son aquellos que se derivan del dominio de aplicación del sistema más que de las necesidades específicas de los usuarios. Se pueden considerar "un complemento" para los Requerimientos de Sistema.

Importante

- Se deben satisfacer para que el sistema funcione correctamente.
- Incluyen terminología especializada del dominio o referencia a conceptos del dominio.
- Pueden restringir los requerimientos existentes o establecer cómo se deben ejecutar dichos requerimientos.

Ejemplo *(sistema para créditos bancarios)*

Regla de Negocio

El cálculo del monto total del crédito debe ser calculado utilizando la fórmula:

Monto Total = cuota mensual x cantidad de meses = (Monto solicitado)/(cantidad de meses)x(tasa del periodo)

El cálculo de la cuota debe ser calculado utilizando la fórmula:

Cuota Mensual = (Monto solicitado)/(cantidad de meses) + Interés mensual

Pruebas de software

Tuesday, April 18, 2023 7:13 PM

Aseguramiento de Calidad: Conceptos

Verificación

En el sentido más general, la verificación es la comprobación de algún evento, proceso, material u otro.

Validación

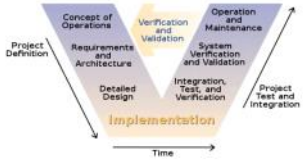
Es la acción de dar fuerza o firmeza a aquello que tiene un peso legal o que es rígido y subsistente

Pruebas de Validación

Las pruebas de validación son el proceso de revisión al que se somete un programa informático para comprobar que cumple con sus especificaciones. El mismo, que suele tener lugar al final de la etapa de desarrollo, se realiza principalmente con la intención de confirmar que la aplicación permita llevar a cabo las tareas que sus potenciales usuarios esperan de ella.

Verificar → ¿Estamos construyendo correctamente el producto?

Validación → ¿Estamos construyendo el producto correcto?



V & V en las etapas de Modelamiento

Nivel de Modelización	Verificación	Validación
Modelo Conceptual		¿Contiene el modelo todos los elementos, sucesos, y relaciones relevantes? ¿Podrá el modelo responder a las cuestiones planteadas?
Modelo Lógico	¿Están los eventos representados correctamente? ¿Son las fórmulas matemáticas y las relaciones correctas? ¿Están las medidas estadísticas formuladas correctamente?	¿Incluye el modelo todos los elementos considerados en el modelo conceptual? ¿Contiene todas las relaciones del modelo conceptual?
Modelo de ordenador/Modelo de Simulación	¿Contiene el código todos los aspectos del modelo lógico? ¿Están las estadísticas y las fórmulas calculadas correctamente? ¿Contiene el modelo errores de codificación?	¿Es el modelo una representación válida del sistema real? ¿Puede el modelo duplicar el comportamiento del sistema real? ¿Es creible el modelo para los expertos del sistema?

Técnicas de Testing como V&V

El Testing consiste en realizar una serie de pruebas controladas sobre el sistema o conjunto de programa. Los dos tipos de prueba más comunes son:

Ascendentes

En estos se comienza por comprobar los módulos básicos y luego se pasa a probar las interrelaciones entre ellos hasta que el modelo ha sido probado como un único sistema.

Descendentes

Las pruebas comienzan con el módulo principal y van bajando hasta los módulos básicos.

Técnicas de Validación

Comparación de salidas entre lo modelado y lo real

- Se podrá aplicar en aquellos casos en los que el sistema exista y se pueda experimentar con él de forma que se obtengan datos de salida del mismo.
- Este método consiste en ejecutar el modelo y obtener una serie de datos de salida y comparar éstos, mediante algún método estadístico, con resultados que se tengan del sistema.

Método de Turing

- Alan Turing sugirió este método como un test de inteligencia artificial.
- En este test, a un experto, o grupo de expertos, se le presentan resúmenes o informes de resultados de ejecución del sistema y del modelo, a los que se les ha dado el mismo formato.
- Estos informes se reparten aleatoriamente a los ingenieros y administradores del sistema, para ver si son capaces de discernir cuáles son los reales del sistema y cuales la imitación resultado de la simulación.
- Si los expertos no son capaces de distinguir entre ambos, se puede concluir que no hay evidencias para considerar inadecuado al modelo. Si descubren diferencias las respuestas sobre lo que encuentran inconsistente se puede utilizar para realizar mejoras en el modelo.

Técnicas de Validación

Comportamientos en casos extremos (test de estrés)

- En algunas ocasiones se puede observar el comportamiento del sistema bajo condiciones extremas.
- Esta es una situación ideal para recoger datos de las medidas de ejecución del sistema real de forma que luego se puedan comparar con los resultados de la simulación, una vez que se ejecute el modelo bajo situaciones similares.
- También es posible que los expertos del sistema puedan predecir el comportamiento del sistema bajo condiciones extremas y utilizar estas predicciones para validar el modelo.

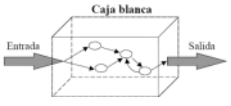
Pruebas de Software

Pruebas unitarias

Es una forma de comprobar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

Para que una prueba unitaria tenga la calidad suficiente se deben cumplir los siguientes requisitos:

- Automatizable → Sin intervención humana (o mínima).
- Completas → Deben cubrir la mayor cantidad de código.
- Repetibles o Reciclables → Las pruebas deben poder ser ejecutadas varias veces.
- Independientes → La ejecución de una prueba no debe afectar a la ejecución de otra.
- Profesionales → Se deben documentar.



Pruebas de Software

Pruebas de Integración

- Son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias.
- Únicamente se refieren a la prueba o pruebas de todos los elementos unitarios que componen un proceso, hecha en conjunto, de una sola vez.
- Sirven para verificar que un gran conjunto de partes de software funcionan juntos.

Pruebas Modulares

- Estas pruebas nos permiten determinar si un modulo del programa esta listo y correctamente terminado.

Pruebas de Aceptación

- Las pruebas de aceptación son pruebas formales que verifican si un sistema satisface los requisitos empresariales.
- Requieren que se esté ejecutando toda la aplicación durante las pruebas y se centran en replicar las conductas de los usuarios.
- Sin embargo, también pueden ir más allá y medir el rendimiento del sistema y rechazar cambios si no se han cumplido determinados objetivos.

Definición de un Plan de Pruebas de SW

El plan de pruebas de software se elabora para atender los objetivos de calidad en un desarrollo de sistemas, encargándose de definir aspectos como por ejemplo los módulos o funcionalidades sujeto de verificación, tipos de pruebas, entornos, recursos asignados, entre otros aspectos.

PASO 1: Analizar los requerimientos de desarrollo de software

- Para elaborar un plan de pruebas de software lo primero es comprender los requerimientos de usuario que componen la iteración (o sistema), que son el sujeto de la verificación de calidad que se va a realizar.
- Se debe analizar toda la información de la ingeniería de requisitos, incluyendo la matriz de trazabilidad (más detalle en **este enlace**), especificaciones y diseño funcional, requisitos no funcionales, casos de uso, historias de usuario (si estás trabajando con metodologías ágiles), entre otra documentación.
- También es muy importante realizar entrevistas con el equipo encargado de la ingeniería de requisitos para aclarar dudas y ampliar la información que sea necesaria.

Definición de un Plan de Pruebas de SW

PASO 2: Identificar las funcionalidades nuevas a probar

- A partir de la documentación del análisis de requisitos y de las entrevistas con el equipo de ingeniería de requisito y desarrollo, debes identificar e incluir en el plan de pruebas de software en la lista de las funcionalidades (si el sistema es nuevo, todo es nuevo)
- En el caso de desarrollos de software integrados a un sistema existente es necesario revisar las funcionalidades que forman parte del desarrollo de software, en todas las capas de la arquitectura.

PASO 3: Identificar las funcionalidades de sistemas existentes que se deben probar

- Se debe identificar las funcionalidades existentes que estén siendo impactadas por el desarrollo de alguna forma, considerando todos los componentes afectados en todas las capas de la arquitectura de software.

Definición de un Plan de Pruebas de SW

PASO 4: Definir la estrategia de pruebas

- Consiste básicamente en seleccionar cuáles son los tipos de pruebas de software que se deben realizar.

PASO 5: Definir criterios de inicio, aceptación y suspensión de pruebas

Criterios de aceptación o rechazo

- Para definir los criterios de aceptación o rechazo, es necesario definir el nivel de tolerancia a fallos de calidad.
- Si la tolerancia a fallos es muy baja se puede definir como criterio de aceptación que el 100% de los casos de prueba estén sin incidencias.
- Lograr este margen en todos los casos de prueba principales y casos bordes será muy difícil, y podría comprometer los plazos del proyecto (incrementa los riesgos), pero asegura la calidad del producto.
- Por otra parte, puede ser que la intención sea realizar un Soft Launch, o un mínimo producto viable, en ese caso se podría definir como criterio de aceptación el 100% de los casos de prueba principales (considerados clave) y 20% de casos de prueba no principales (casos borde).

Una vez logradas las condiciones, se darán por aceptadas las pruebas y el desarrollo de software.

Definición de un Plan de Pruebas de SW

Criterios de inicio o reanudación

- Definen las condiciones que se deben cumplir para dar inicio o reanudar las pruebas.
- Por ejemplo, en el caso de inicio la condición podría ser la instalación de los componentes de software en el ambiente y que los casos de pruebas de verificación de ambiente sean exitosos.
- Para el caso de la reanudación las condiciones están relacionadas, se determina a partir de cuales criterios de suspensión se presentaron para detener las pruebas. Una vez que estás condiciones ya no existan (sean solventadas) se procede con la reanudación.

Criterios de suspensión

- Las condiciones van a depender de los acuerdos de nivel de servicio (SLA) internos de la organización y también de los acuerdos establecidos en cada proyecto individual.
- Por ejemplo, si se tiene un equipo de pruebas que comparte su esfuerzo entre varios proyectos, se puede definir un criterio de suspensión exigente, un determinado porcentaje de casos fallidos que resulten en incidencias. Si la condición se cumple, se detienen las pruebas y se dedica el personal a otras actividades.
- Por otra parte si se tiene un equipo de pruebas con personal dedicado, el criterio de suspensión puede ser poco exigente, por ejemplo solo ocurriendo si se bloquean por incidencia todos los casos de prueba.