LOS CASOS DE USO PARA LA GENERACION DE CASOS DE PRUEBA ENTESTING DE SOFTWARE

Ing. Freddy Rocabado Ibáñez
rocosfree@gmail.com
Docente Ingeniería Informática
Universidad Nacional "Siglo XX"
Llallagua, Bolivia.

Resumen - *En ingeniería del software, un caso de uso es una técnica para la captura de requisitos funcionales de un nuevo sistema o una actualización de software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico. Normalmente, en los casos de usos se evita el empleo de jergas técnicas, prefiriendo en su lugar un lenguaje más cercano al usuario final. En ocasiones, se utiliza a usuarios sin experiencia junto a los analistas para el desarrollo de casos de uso.*

En otras palabras, un caso de uso es una secuencia de interacciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que inicia un actor principal sobre el propio sistema. Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema.

Una relación es una conexión entre los elementos del modelo, por ejemplo, la especialización y la generalización son relaciones. Los diagramas de casos de uso se utilizan para ilustrar los requerimientos del sistema al mostrar cómo reacciona a eventos que se producen en su ámbito o en él mismo. De esta forma se tendrá la capacidad de crear casos de prueba a partir de los casos de uso realizando la traza de unos a otros y esto representa una habilidad vital para asegurar un producto de calidad.

Palabras Claves - *, Caso de Uso, Casos de Prueba, Ingeniería de software Prueba, Prueba de Software.*

Abstract y Keywords - *In software engineering, a use case is a technique for capturing functional requirements of a new system or softwareupdate. Each use case provides one or more scenarios that indicate how the system should interact with the user or with another system to achieve a specific goal. Normally, in use cases the use of technical jargon is avoided, preferring instead alanguage closer to the end user. Sometimes inexperienced users are used alongside analysts to develop use cases.*
In other words, a use case is a sequence of interactions that will develop between a system and its actors in response to an event initiated by a main actor on the system itself. Use case diagrams serve to specify the communication and behavior of a system through its interaction with users and/or other systems. Or what is the same, a diagram that shows the relationship between the actors and the use cases in a system.
A relationship is a connection between the elements of the model, for example specialization and generalization are relationships. Use case diagrams are used to illustrate system requirements by showing how the system reacts to events that occur within or within its scope. In this way, you will have the ability to create test cases from the use cases, tracing each other and this represents a vital skill to ensure a quality product.

Keywords - *Software engineering, Software Testing, Testing, Test Case, Use Case.*

.

1. INTRODUCCIÓN

En esencia, un caso de uso narra una historia estilizada sobre cómo interactúa un usuario final (que tiene cierto número de roles posibles) con el sistema en circunstancias específicas. La historia puede ser un texto narrativo, un lineamiento de tareas o interacciones, una descripción basada en un formato o una representación diagramática. Sin importar su forma, un caso de uso ilustra el software o sistema desde el punto de vista del usuario final.

El primer paso para escribir un caso de uso es definir un conjunto de "actores" que estarán involucrados en la historia. Los actores son las distintas personas (o dispositivos) que usan el sistema o producto en el contexto de la función y comportamiento que va a describirse. Los actores representan los papeles que desempeñan las personas (o dispositivos) cuando opera el sistema. Con una definición más formal, un actor es cualquier cosa que se comunique con el sistema o producto y que sea externo a éste. Todo actor tiene uno o más objetivos cuando utiliza el sistema.

Es necesario escribir los requerimientos con diferentes niveles de detalle, ya que varios lectores los usarán de distintas formas como también los usuarios. De éstos, los primeros por lo general no están interesados en la manera en que se implementará el sistema, y quizá sean administradores a quienes no les atraigan las facilidades detalladas del sistema. Mientras que los segundos necesitan conocer con más precisión qué hará el sistema, ya que están preocupados sobre cómo apoyará los procesos de negocios o porque están inmersos en la implementación del sistema.

Para la mayoría de los sistemas grandes, todavía se presenta una fase de ingeniería de requerimientos claramente identificable, antes de comenzar la implementación del sistema. El resultado es un documento de requerimientos que puede formar parte del contrato de desarrollo del sistema. Desde luego, por lo común hay cambios posteriores a los requerimientos, en tanto que los requerimientos del usuario podrían extenderse como requerimientos de sistema más detallados. Sin embargo, el enfoque ágil para alcanzar, al mismo tiempo, los requerimientos a medida que el sistema se desarrolla rara vez se utilizan en el diseño de sistemas grandes.

Un caso de prueba es uno de los resultados de las pruebas de software. Los casos de prueba se definen como el conjunto de variables y condiciones utilizadas para verificar una determinada característica o funcionalidad del software. Siguiendo los pasos de la prueba, el ingeniero de control de calidad puede comparar los resultados reales y los esperados de la prueba para ver si el software se comporta según la intención inicial. La descripción anterior simplifica un caso de prueba a su esencia principal, sin embargo, si alguien decide buscar ejemplos de casos de prueba, digamos, ejemplos de casos de prueba para aplicaciones web o ejemplos de casos de prueba funcionales, los resultados pueden ser algo sorprendentes. A menudo, nos encontramos con casos de prueba que tienen un formato y una estructura diferentes y que carecen de partes cruciales.

Como definición básica del contenido, hay que mencionar, en primer lugar, que testing es el proceso de ejecutar un conjunto de elementos software con el fin de encontrar errores. Por tanto, testear no es demostrar que no hay errores en el programa ni únicamente mostrar que el programa funciona correctamente, ambas son definiciones incorrectas y, sin embargo, comúnmente utilizadas.

Así, un error y un defecto software (y como consecuencia un fallo) existen cuando el software no hace lo que el usuario espera que haga, es decir, aquello que se ha acordado previamente en la especificación de requisitos. En una gran parte de los casos, esto se produce por un error de comunicación con el usuario durante la fase de análisis de requisitos o por un error de codificación.

2. DESARROLLO

2.1. Modelo de Casos de Uso

Un caso de uso provee a los desarrolladores un panorama sobre lo que desean los usuarios. Está libre de detalles técnicos o de implementación. Podemos pensar en un caso de uso como una secuencia de transacciones en un sistema. El modelo de casos de uso se basa en las interacciones y relaciones de los casos de uso individuales. Un caso de uso siempre describe tres cosas: un actor que inicia un evento, el evento que desencadena un caso de uso y el caso de uso que realiza las acciones desencadenadas por el evento. En un caso de uso, un actor que utiliza el sistema inicia un evento que a su vez genera una serie relacionada de interacciones en el sistema.

Los casos de uso se utilizan para documentar una transacción o evento individual. Se introduce un evento en el sistema, el cual ocurre en un tiempo y lugar específicos para provocar que el sistema haga algo. Los casos de uso son una técnica para la especificación de requisitos funcionales propuesta inicialmente por Ivar Jacobson [Jacobson, 1987], [Jacobson et al. 1992] e incorporada a UML donde modela la funcionalidad del sistema tal como la perciben los agentes externos, denominados actores, que interactúan con el sistema desde un punto de vista particular. Sus componentes principales son:

- Sujeto: sistema que se modela
- Casos de uso: unidades funcionales completas
- Actores: entidades externas que interactúan con el sistema

El sujeto se muestra como una caja negra que proporciona los casos de uso.

2.2. Casos de Uso

Un caso de uso se define como un conjunto de acciones realizadas por el sistema que dan lugar a un resultado observable. El caso de uso especifica un comportamiento que el sujeto puede realizar en colaboración con uno o más actores, pero sin hacer referencia a su estructura interna. El caso de uso puede contener posibles variaciones de su comportamiento básico incluyendo manejo de errores y excepciones donde una instanciación de un caso de uso es un escenario que representa un uso particular del sistema (un camino) Características de los casos de uso:

• Un caso de uso se inicia por un actor.
• Los casos de uso proporcionan valores a los actores.
• La funcionalidad de un caso de uso debe ser completa.

2.3. Formatos de Casos de Uso

Formato de Caja Negra. Se enfoca en la responsabilidad que tiene el caso de uso, no en cómo la realiza
Formato de Caja Blanca. Indica qué debe hacer el caso de uso (Responsabilidad) y cómo realizarla.

Actor	Nombre del actor
Casos de Uso	Nombre de los casos de uso en la que participa
Tipo	Principal, Apoyo o Pasivo
Descripción	Breve Descripción del Actor

Grafico 1. Documentación de Actores
Fuente.https://academicos.azc.uam.mx/jfg/diapositivas/adsi/Unidad_5.pdf

Documentación de Caso de Uso

Caso de Uso	Nombre del caso de uso
Actores	Actores involucrados en el caso de uso
Tipo	Básico, Extensión, Inclusión, Generalización
Propósito	Objetivo del caso de uso
Pre condiciones	Lo que debe cumplirse antes de que se ejecute el caso de uso
Post condiciones	Lo que debe ocurrir cuando el caso de uso se ejecute de manera correcta
Escenario de éxito	Flujo de eventos principal suponiendo que no ocurren errores
Extensiones	Flujos alternos que se presentan en el caso de uso, posiblemente debido a errores

Gráfico 2. Documentación de Casos de Uso
Fuente.https://academicos.azc.uam.mx/jfg/diapositivas/adsi/Unidad_5.pdf

La Creación del Modelo de Casos de Uso deberá seguir el siguiente orden.

- Los nombres de los casos de uso deben ser nombres de acciones
- Se deben identificar las relaciones existentes entre los distintos casos de uso (extensión, inclusión, agregación)
- Se deben identificar las relaciones entre los casos de uso y los actores
- No puede haber comunicación directa entre los actores

2.4. Normas de aplicación

Los casos de uso evitan típicamente el lenguaje técnico, prefiriendo la lengua del usuario final o del experto del campo del saber al que se va a aplicar. Los casos del uso son a menudo elaborados en colaboración por los analistas de requisitos y los clientes.

Cada caso de uso se centra en describir cómo alcanzar una única meta o tarea. Desde una perspectiva tradicional de la ingeniería de software, un caso de uso describe una característica del sistema. Para la mayoría de proyectos de software, esto significa que quizás a veces es necesario especificar decenas o centenares de casos de uso para definir completamente el nuevo sistema. El grado de la formalidad de un proyecto particular del software y de la etapa del proyecto influenciará el nivel del detalle requerido en cada caso de uso.

Los casos de uso pretenden ser herramientas simples para describir el comportamiento del software o de los sistemas. Un caso de uso contiene una descripción textual de todas las maneras que los actores previstos podrían trabajar con el software o el sistema. Los casos de uso no describen ninguna funcionalidad interna (oculta al exterior) del sistema, ni explican cómo se implementará. Simplemente muestran lo que el actor hace o debe hacer para realizar una operación.

Un caso de uso debe:

- Describir una tarea del negocio que sirva a una meta de negocio.
- Tener un nivel apropiado del detalle.
- Ser bastante sencillo como para que un desarrollador lo elabore en un único lanzamiento.

Situaciones que pueden darse:

- Un actor se comunica con un caso de uso (si se trata de un actor primario la comunicación la iniciará el actor, en cambio si es secundario, el sistema será el que inicie la comunicación).
- Un caso de uso extiende otro caso de uso.
- Un caso de uso utiliza otro caso de uso.

2.5. Facilidades

La técnica de casos de uso tiene éxito en sistemas interactivos, ya que expresa la intención que tiene el actor (su usuario) al hacer uso del sistema. Como técnica de extracción de requisito permite que el analista se centre en las necesidades del usuario, qué espera este lograr al utilizar el sistema, evitando que la gente especializada en informática dirija la funcionalidad del nuevo sistema basándose solamente en criterios tecnológicos.

A su vez, durante el análisis de software, el analista se concentra en las tareas centrales del usuario describiendo por lo tanto los casos de uso que mayor valor aportan al negocio. Esto facilita luego la priorización del requisito. Aunque comúnmente se asocian a la fase de Test de una aplicación, esta idea es errónea, y su uso se extiende mayormente a las primeras fases de un desarrollo.

2.6. Limitaciones

Los casos de uso pueden ser útiles para establecer requisitos de comportamiento, pero no establecen completamente los requisitos funcionales ni permiten determinar los requisitos no funcionales. Los casos de uso deben complementarse con información adicional como reglas de negocio, requisitos no funcionales, diccionario de datos que complementen los requisitos del sistema como también los casos de prueba. Sin embargo, la ingeniería del funcionamiento especifica que cada caso crítico del uso debe tener un requisito no funcional centrado en el funcionamiento asociado.

3.1. Casos de Prueba

Todo el mundo desde la NASA y TESLA a corporaciones de nivel empresarial pueden beneficiarse de los casos de prueba y que estos operan de la mejor manera. Escribir casos de prueba excelentes es solo una forma más de mejorar la eficiencia y la eficacia en el trabajo en equipo de los testeadores de software.

Un caso de prueba es uno de los resultados de las pruebas de software. Los casos de prueba se definen como el conjunto de variables y condiciones utilizadas para verificar una determinada característica o funcionalidad del software. Siguiendo los pasos de la prueba, el ingeniero de control de calidad puede comparar los resultados reales y los esperados de la prueba para ver si el software se comporta según la intención inicial.

La descripción anterior simplifica un caso de prueba a su esencia principal, sin embargo, si alguien decide buscar ejemplos de casos de prueba, digamos, ejemplos de casos de prueba para aplicaciones web o ejemplos de casos de prueba funcionales, los resultados pueden ser algo sorprendentes. A menudo, nos encontramos con casos de prueba que tienen un formato y una estructura diferentes y que carecen de partes cruciales.

Como regla de oro a la hora de redactar casos de prueba, hay que respetar la siguiente estructura:

Paso 1: ID de caso de prueba.
Todos los casos de prueba deben llevar ID únicos para representarlos. En la mayoría de los casos, seguir una convención para este ID de nomenclatura ayuda con la organización, la claridad y la comprensión.
Paso 2: Descripción de la prueba.
Esta descripción debe detallar qué unidad, característica o función se está probando o qué se está verificando.
Paso 3: Supuestos y condiciones previas.
Esto implica que se cumplan las condiciones antes de la ejecución del caso de prueba. Un ejemplo sería requerir una cuenta de Outlook válida para iniciar sesión.
Paso 4: Datos de prueba.
Esto se relaciona con las variables y sus valores en el caso de prueba. En el ejemplo de un inicio de sesión por correo electrónico, sería el nombre de usuario y la contraseña de la cuenta.
Paso 5: Pasos a ejecutar.
Estos deben ser pasos fácilmente repetibles ejecutados desde la perspectiva del usuario final. Por ejemplo, un caso de prueba para iniciar sesión en un servidor de correo electrónico podría incluir estos pasos:
- Abra la página web del servidor de correo electrónico.
- Introduzca su nombre de usuario.
- Introducir la contraseña.
- Haga clic en el botón "Entrar" o "Iniciar sesión".

Paso 6: Resultado Esperado.
Esto indica el resultado esperado después de la ejecución del paso del caso de prueba. Al ingresar la información de inicio de sesión correcta, el resultado esperado sería un inicio de sesión exitoso.
Paso 7: Resultado real y condiciones posteriores.
En comparación con el resultado esperado, podemos determinar el estado del caso de prueba. En el caso del inicio de sesión por correo electrónico, el usuario iniciará sesión correctamente o no. La condición posterior es lo que sucede como resultado de la ejecución del paso, como ser redirigido a la bandeja de entrada del correo electrónico.
Paso 8: Contraseña errónea
La determinación del estado de aprobado / reprobado depende de cómo se comparan entre sí el resultado esperado y el resultado real.
Mismo resultado = Aprobado
Diferentes resultados = Falla

El objetivo principal de cualquier caso de prueba es proporcionar una guía clara y concisa paso a paso, una “escalera hacia los errores” que proporcione una orientación clara de lo que ha salido mal.

3.2. ¿Cómo redactar casos de prueba?

Para aprender a escribir casos de prueba hay que prestar mucha atención a los detalles y conocer bien la aplicación que se está probando. Vamos a ilustrar esto con un ejemplo de caso de prueba para una página de inicio de sesión. La elección de este tipo concreto para demostrar nuestros métodos comprobados viene dictada por el altísimo valor de la seguridad y su funcionalidad habitualmente limitada. De ahí la relativa disponibilidad para los probadores de todos los niveles.

3.3. ¿Cuáles son las mejores prácticas para redactar casos de prueba de calidad?

La forma de escribir pruebas y casos de prueba eficaces se puede optimizar con el tiempo. Algunos y las mejores prácticas incluya el uso de títulos sólidos, descripciones sólidas y mantener el lenguaje conciso y claro.

Pero también querrá incluir condiciones previas, suposiciones y los resultados esperados. Toda esta información es relevante para el probador de software, especialmente cuando se determina si el caso de prueba debe ser un "aprobado" o un "error" en su lugar.

Una hoja de trucos para crear casos de prueba que funcionen bien es la siguiente:

- Mantenga las cosas simples y transparentes.
- Haga que los casos de prueba sean reutilizables.
- Mantenga únicos los ID de los casos de prueba.
- La revisión por pares es importante.
- Los casos de prueba deben tener en cuenta al usuario final o los requisitos definidos.
- Especifique los resultados esperados y los supuestos.

3.4. ¿Qué es una suite de pruebas?

Un conjunto de pruebas entra en juego para los casos de prueba en lo que se refiere al código fuente, la colección de dependencias o el conjunto de pruebas que se realizarán en el código. Los conjuntos de pruebas le permiten categorizar los casos de prueba de manera que se alineen con cualquier análisis o necesidad de planificación.

Esto significa que las funciones principales del software pueden tener su propio conjunto de pruebas, mientras que otro conjunto de pruebas es para un tipo de prueba específico, como humo o seguridad. Piense en los conjuntos de pruebas como una estantería para organizar sus casos de prueba.

3.5. ¿Qué es un plan de prueba?

Por el contrario, un plan de prueba se parece más al paraguas que cubre todas las suites de prueba. Si los casos de prueba son libros y los conjuntos de pruebas son estanterías, los planes de prueba son la sala que contiene la estantería.
Generalmente, los planes de prueba se configuran en términos de pruebas manuales, pruebas automatizadas y un formato general de cómo realizar las pruebas. Probarán el software desde la base utilizando conjuntos de pruebas y casos de prueba antes de implementar cambios o agregar nuevas funciones.

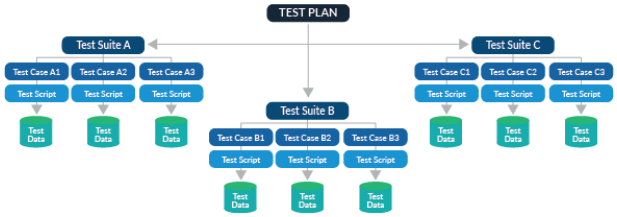

Gráfico 3. Plan de Pruebas de software
Fuente: https://es.parasoft.com/blog/how-to-write-test-cases-for-software-examples-tutorial/

4.1. Testing de Software

En el proceso de desarrollo de software es normal encontrar errores. Cuando esto sucede en la etapa de testing o prueba de software, no supone un gran inconveniente. Continuar sin abordarlos puede generar problemas graves para todas las partes involucradas en el proceso de desarrollo del proyecto. El testing de software juega un papel fundamental y supone una garantía de calidad de suma importancia para cualquier empresa.

El testing de software o software QA, es un proceso para verificar y validar la funcionalidad de un programa o una aplicación de software con el objetivo de garantizar que el producto de software esté libre de defectos. La intención final es que coincida con los requisitos esperados para entregar un producto de calidad. Implica la ejecución de componentes de software o sistema utilizando herramientas manuales o automatizadas para evaluar una o más propiedades de interés.

El testing de software es un proceso paralelo al desarrollo de software cuyas tareas deben ir realizándose a medida que se construye el producto para evitar problemas en la funcionalidad de manera previa a su lanzamiento.

4.2. Verificación y validación

El testing de software pertenece a una actividad o etapa del proceso de producción de software denominada Verificación y Validación, usualmente abreviada como V&V es el nombre genérico dado a las actividades de comprobación que aseguran que el software respeta su especificación de requerimiento en la etapa de identificación de los casos de uso y satisface las necesidades de sus usuarios. El sistema debe ser verificado y validado en cada etapa del proceso de desarrollo utilizando los documentos (descripciones) producidas durante las etapas anteriores.

En rigor no solo el código debe ser sometido a actividades de V&V sino también todos los subproductos generados durante el desarrollo del software. En efecto, estos subproductos deben ser verificados de forma tal de que exista mayor confianza en que cumplen con los requerimientos del cliente en particular el equipo de desarrollo debe asegurarse de que la arquitectura podrá incorporar los cambios previstos a bajo costo y que esta habilita otras propiedades que haya solicitado el cliente tales como seguridad, confiabilidad, portabilidad, etc.

Si bien estos términos en su uso cotidiano pueden llegar a ser sinónimos, en Ingeniería de Software tienen significados diferentes y cada uno tiene una definición más o menos precisa.

-Validación: ¿estamos construyendo el producto correcto?
-Verificación: ¿estamos construyendo el producto correctamente?

En este sentido, la verificación consiste en corroborar que el programa respeta su especificación, mientras que validación significa corroborar que el programa satisface las expectativas del usuario.

En otras palabras, la verificación es una actividad desarrollada por ingenieros teniendo en cuenta un modelo del programa y el programa en sí, en tanto que la validación la debe realizar el usuario teniendo en cuenta lo que él espera del programa y el programa en sí. Existen varias técnicas dentro del marco de la V&V desde las más informales (prototipación de requerimientos, revisión de requerimientos, casos de uso, etc.), pasando por las semiformales (el testing es la más conocida pero ciertas técnicas de análisis estático de código se usan frecuentemente), hasta la prueba formal de programas, el cálculo de refinamiento, etc.

4.3. El proceso de testing

Es casi imposible, excepto para los programas más pequeños, testear un software como si fuera una única entidad monolítica. Como hemos visto en los capítulos sobre arquitectura y diseño, los grandes sistemas de software están compuestos de subsistemas, módulos y subrutinas. Se sugiere, por lo tanto, que el proceso de testing esté guiado por dicha estructura, como muestra la Figura 4. Más aun, si el proceso sugerido se combina adecuadamente con el proceso de desarrollo, como muestra la Figura 6, entonces se habilita la posibilidad de ir detectando errores de implementación lo más tempranamente posible lo que a su vez reduce el costo de desarrollo.

Como sugiere la Figura 4, un sistema complejo suele testearse en varias etapas que por lo general se ejecutan siguiendo una estrategia, aunque el proceso general es iterativo. Se debe volver a las fases anteriores cada vez que se encuentra un error en la fase que está siendo ejecutada. Por ejemplo, si se encuentra un error durante el testing de un subsistema particular, una vez que aquel haya sido reparado se deberán testear la unidad donde estaba el error y luego el módulo al cual pertenece la unidad reparada.

En general, una vez detectado un error se sigue el proceso graficado en la Figura 5. De aquí que a las iteraciones del proceso de la Figura 4 se las llame re-testing. Dado que no hay una definición precisa de subsistema e incluso de unidad, el proceso de testing sugerido debe ser considerado como una guía que debe ser adaptada a cada caso específico.

El testing de aceptación mencionado en la figura pertenece a la validación de un sistema (y no a la verificación) dado que es el usuario el que usa el sistema en un entorno más o menos real con el fin de comprobar si es una implementación razonable de los requerimientos. En la práctica industrial usualmente se entiende que el testing es una actividad que se realiza una vez que los programadores han terminado de codificar; en general es sinónimo de testing de aceptación. Entendido de esta forma el testing se convierte en una actividad costosa e ineficiente desde varios puntos de vista:

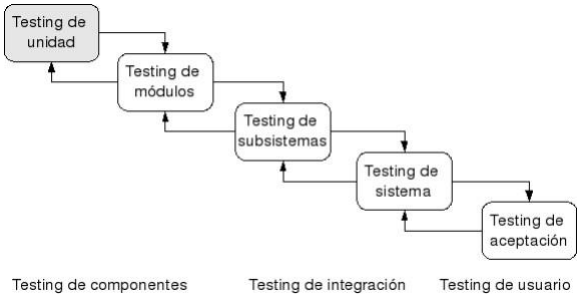

Figura 4: El proceso general de testing (fuente [1]).
Fuente: https://www.fceia.unr.edu.ar/ingsoft/testing-intro-a.pdf

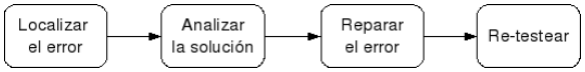

Figura 5: El proceso de debugging (fuente [1]).
Fuente: https://www.fceia.unr.edu.ar/ingsoft/testing-intro-a.pdf

Los testers estarán ociosos durante la mayor parte del proyecto y estarán sobrecargados de trabajo cuando este esté por finalizar. Los errores tienden a ser detectados muy tarde. Se descubre un gran número de errores cuando el presupuesto se está terminando. Los errores tienden a ser detectados por los usuarios y no por el personal de desarrollo lo que implica un desprestigio para el grupo de desarrollo.

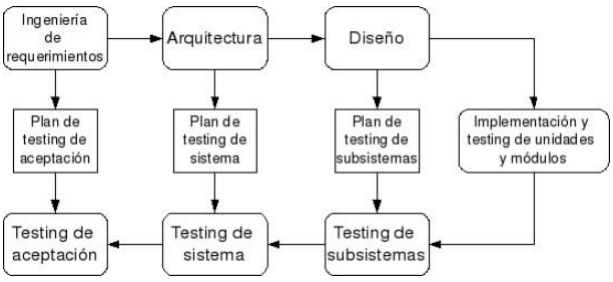

Figura 6: Coordinación entre el proceso de testing y el proceso de desarrollo (adaptado de [1])
Fuente: https://www.fceia.unr.edu.ar/ingsoft/testing-intro-a.pdf

Por lo tanto, se sugiere un proceso de testing mejor imbricado con el proceso general de desarrollo, como se muestra en la Figura 6. En cuanto el equipo de desarrollo cuenta con un documento de requerimientos más o menos estable, los testers pueden comenzar a definir casos de prueba para el testing de aceptación, dado que este se basa en validar los requerimientos. De la misma forma, tan pronto como se ha definido la arquitectura o el diseño del sistema, los testers pueden usar dicha documentación para

calcular casos de prueba para el testing de subsistemas y del sistema.

4.4. Por qué es importante el Testing de software

Pocos pueden argumentar en contra de la necesidad de un control de calidad al desarrollar software. Los retrasos en las entregas o los defectos del software pueden dañar la reputación de una marca, lo que provoca la frustración y la pérdida de clientes. En casos extremos, un error o defecto puede degradar los sistemas interconectados o causar fallas graves.

Aunque el testing de software cuestan dinero, las empresas pueden ahorrar millones por año en desarrollo y soporte si cuentan con una buena técnica de prueba y procesos de control de calidad. Las primeras pruebas de software descubren problemas antes de que un producto salga al mercado. Cuanto antes reciban los equipos de desarrollo los comentarios de las pruebas, antes podrán abordar problemas como:

- Defectos arquitectónicos
- Malas decisiones de diseño
- Funcionalidad no válida o incorrecta
- Vulnerabilidades de seguridad
- Problemas de escalabilidad

Cuando el desarrollo deja un amplio espacio para las pruebas, mejora la confiabilidad del software y las aplicaciones de alta calidad se entregan con pocos errores. Un sistema que cumple o incluso supera las expectativas del cliente genera potencialmente más ventas y una mayor cuota de mercado.

4.5. Mejores Prácticas de Testing de Software

El testing de software siguen un proceso común, las tareas o pasos incluyen la definición del entorno de prueba, el desarrollo de casos de prueba, la escritura de guiones, el análisis de los resultados de la prueba y el envío de informes de defectos. Las pruebas pueden llevar mucho tiempo, las pruebas manuales o bajo demanda pueden ser suficientes para compilaciones pequeñas. Sin embargo, para sistemas más grandes, las herramientas se utilizan con frecuencia para automatizar tareas. El testing automatizado ayudan a los equipos a implementar diferentes escenarios, probar diferenciadores (como mover componentes a un entorno de nube) y obtener comentarios rápidamente sobre lo que funciona y lo que no.

Un buen enfoque de prueba abarca la interfaz de programación de aplicaciones, la interfaz de usuario y los niveles del sistema. Además, cuantas más pruebas se automaticen y se ejecuten antes, mejor. Algunos equipos desarrollan herramientas de automatización de pruebas internas. Sin embargo, las soluciones de los proveedores ofrecen funciones que pueden optimizar las tareas clave de gestión de pruebas, como:

Prueba continua: los equipos de proyecto prueban cada compilación a medida que está disponible. Este tipo de testing de software se basa en la automatización de pruebas que se integra con el proceso de implementación. Permite que el software se valide en entornos de prueba realistas en una etapa más temprana del proceso, lo que mejora el diseño y reduce los riesgos.

Gestión de la configuración: las organizaciones mantienen de forma centralizada los activos de testing y realizan un seguimiento de las compilaciones de software para probar. Los equipos obtienen acceso a activos como código, requisitos, documentos de diseño, modelos, scripts de prueba y resultados de prueba. Los buenos sistemas incluyen autenticación de usuarios y seguimientos de auditoría para ayudar a los equipos a cumplir con los requisitos de conformidad con un mínimo esfuerzo administrativo.

Virtualización de servicios: es posible que los entornos de prueba no estén disponibles, especialmente en las primeras etapas del desarrollo del código. La virtualización de servicios simula los servicios y sistemas que faltan o que aún no se han completado, lo que permite a los equipos reducir las dependencias y realizar el testing antes. Pueden reutilizar, implementar y cambiar una configuración para probar diferentes escenarios sin tener que modificar el entorno original.

Seguimiento de defectos o errores: la supervisión de defectos es importante tanto para los equipos de testing como para los de desarrollo para medir y mejorar la calidad. Las herramientas automatizadas permiten a los equipos realizar un seguimiento de los defectos, medir su alcance e impacto y descubrir problemas relacionados.

Métricas e informes: los informes y la analítica permiten a los miembros del equipo compartir el estado, los objetivos y los resultados de las pruebas. Las herramientas avanzadas integran las métricas del proyecto y presentan los resultados en un panel. Los equipos ven rápidamente el estado general de un proyecto y pueden supervisar las relaciones entre las pruebas, el desarrollo y otros elementos del proyecto

5. Conclusiones.

Identificando y realizando de manera correcta los casos de uso, estas se contrastan con los casos de prueba de esa manera se va asegurando los requisitos de la funcionalidad del software previstos con anterioridad.

Utilizando los casos de uso para generar casos de prueba apoya al equipo de testing a comenzar el trabajo en el ciclo de desarrollo del software permitiéndoles identificar y corregir los defectos que pueden ser muy costosos de

remediar en lo que se denomina las iteraciones posteriores, esto con la verificación y validación de los mismos.

Además, siguiendo sistemáticamente todo lo anterior dicho se asegura en tiempo la entrega del producto software confiable a los testers y desarrolladores les permite simplificar el proceso de prueba y garantizando que se haya hecho todas las pruebas necesarias para que el producto software sea entregado de la mejor forma posible.

Con esta forma de trabajo de contraste de los casos de uso y los casos de prueba se encuentra una coordinación entre desarrollo y testing de software adecuado siguiendo una planificación y ejecución del proyecto.

5. Referencia

1.https://www.javier8a.com/itc/bd1/ldIngenieria.de.software.enfoque.practico.7ed.Pressman.PDF2
2.https://gc.scalahed.com/recursos/files/r161r/w25469w/ingdelsoftwarelibro9_compressed.pdf
3. https://qawerk.es/blog/como-redactar-casos-de-prueba/
4. Bolaños.D, Sierra A, Alarcon M, (2017) Pruebas de Software y JUnit Un análisis en profundidad y ejemplos prácticos, PEARSON EDUCACIÓN, S.A., Madrid
5.https://repositorio.grial.eu/bitstream/grial/1155/1/UML%20-%20Casos%20de%20uso.pdf5
6. https://academicos.azc.uam.mx/jfg/diapositivas/adsi/Unidad_5.pdf
7. https://es.wikipedia.org/wiki/Caso_de_uso
8. https://qawerk.es/blog/como-redactar-casos-de-prueba/
9.https://es.parasoft.com/blog/how-to-write-test-cases-for-software-examples-tutorial/
10. https://profile.es/blog/que-es-el-testing-de-software/
11. https://www.fceia.unr.edu.ar/ingsoft/testing-intro-a.pdf
12. https://www.ibm.com/es-es/topics/software-testing