CU vs US

Ambos conceptos, casos de uso e historias de usuarios, sirven para presentar información sobre lo que va a hacer el sistema, pero mantienen ciertas diferencias entre sí que los hacen tener un enfoque especial dependiendo del uso que se le quiera dar.

En primera instancia, las historias de usuario sirven para recopilar un conjunto de relatos cortos, que quepan en un párrafo, de usuarios que mencionen qué desean que haga el sistema y para qué, es decir, qué es lo que obtendrían de que el sistema de comporte de tal manera.

Por su parte, los casos de uso organizan los requerimientos a forma de interacciones que tiene el usuario con el sistema y todo lo que debería ser capaz de hacer, desde el punto de vista del usuario.

Se diferencian entre sí en cuanto al lenguaje que se utiliza, puesto que las historias de usuario pueden ser algo casual, mientras que casos usualmente de uso se expresa de forma más detallada y lineal, además de que las historias te muestran el por qué detrás de cada requerimiento, mientras que los casos se centran en la interacción del usuario con el sistema sin describir la razón detrás de esas interacciones.

Es recomendable utilizar las historias de usuario en casos donde se utilice un modelo de desarrollo ágil, basándose en la colaboración, interacción directa con los usuarios y se está abierto al cambio constante. Por otro lado, se recomienda utilizar casos de uso cuando se esté en un proceso de desarrollo tradicional altamente controlado y con una especificación detallada en el que las historias de usuario resulten vagas en descripción a comparación.

Habilidades para requerimientos

La IEEE denomina a los requerimientos como la condición o capacidad que debe satisfacer o poseer un sistema o una componente de un sistema para satisfacer un contrato, un standard, una especificación u otro documento formalmente

impuesto. Por su parte, la ingeniería de requerimientos se basa en el descubrimiento, refinamiento, modelización, especificación y validación de los requerimientos, siendo indispensable en esta etapa que exista una correcta comunicación entre el cliente y el desarrollador.

Para esto, todo aquel que se vea en la tarea de transformar las ideas del cliente a una realidad tendrá que pasar por este proceso y deberá tener las habilidades técnicas necesarias para llegar a un entendimiento de los requerimientos y las complicaciones, restricciones y coherencia entre ellos.

- **Capacidad de abstracción:** Debe ser capaz de comprender la idea del cliente y transformar ese concepto abstracto/ambiguo en algo objetivo y conciso, que no dé lugar a confundirla con interpretaciones erróneas.
- **Habilidades comunicativas:** Se debe poder comunicar con el cliente de modo que se llegue a un punto de entendimiento sobre qué es exactamente lo que quiere, además de poder negociar y hacerle saber de forma clara y exacta si alguno de los requerimientos que está planteando no es posible ni/o necesario para el proyecto en cuestión.
- **Capacidad de análisis:** Saber priorizar e identificar si es que los requerimientos son coherentes entre sí, si son necesarios, si son claros de entender, si es que puede llegar a cambiar en una etapa tardía de desarrollo o si son repetitivos.

La importancia de estas tres habilidades finalmente recae en que queden lo más claras posibles las ideas del cliente, pues en ellas está la base de todo el proyecto y si no se llegaran a registrar correctamente podría significar un retraso en el desarrollo o incluso el fracaso de este, siendo que se estima que el 53% de los proyectos de software fracasan por no por no realizar un análisis previo/eficiente de los requisitos.

Métodos y Técnicas de Pruebas

La etapa de pruebas es una parte esencial en todo proceso de desarrollo, puesto que de ella se puede verificar el estado del proyecto sobre varios aspectos como

funcionalidad, rendimiento, usabilidad (UI/experiencia de usuario) y seguridad, por mencionar algunos.

Esta etapa generalmente se implementa al final del desarrollo en modelos lineales, como el de cascada, esperando hasta que se termine de desarrollar casi por completo el proyecto para poder probarlo y a partir de ahí realizar el mantenimiento necesario.

En este caso, me centraré en el modelo de ciclo de vida del software en espiral, el cual no solo tiene su etapa de pruebas en un punto temprano del proyecto, sino que estas pruebas se repiten constantemente a lo largo de todo el desarrollo sin tener que esperar a la finalización del producto final.

Estas pruebas ocurren en la tercera etapa en un ciclo de 4 etapas, y se llevan a cabo con prototipos que son desarrollados en base a un previo entendimiento de los requerimientos y un análisis de riesgo, por lo que los resultados de estas pruebas son realmente útiles para determinar el estado del proyecto e ir trabajando en las áreas que necesiten mayor atención entrando en la cuarta fase, que se basa en una planeación sobre cuál será el siguiente paso antes de repetir el ciclo de nuevo, hasta que se esté lo suficientemente satisfecho para poder llegar al producto final.

La utilidad de implementar la etapa de pruebas de esa manera es que se estará proporcionado información sobre las necesidades y carencias del proyecto continuamente, permitiendo que sea pulido con cada vuelta y convirtiéndolo en una tarea recurrente, más que tan solo parte de los pasos finales del desarrollo, cuando ya se invirtió una gran cantidad de recursos y no existe la posibilidad de revertir lo que se ha hecho.

Mantenimiento de Software como Competencia

El mantenimiento de software es incluido como una competencia que se supone que debería tener todo egresado de la licenciatura en ingeniería de software, estipulando que esta forma parte de aquel que: mantiene productos de software heredados en diferentes dominios de aplicación, optimizando los recursos

humanos, materiales, económicos y de tiempo, y atendiendo las necesidades de la organización. Pero ¿es correcto el incluirlo como una competencia?

Para empezar ¿qué es exactamente una competencia para el plan de estudios? En un documento institucional a modo de introducción al modelo educativo MEFI utilizado por la UADY, se explica que una competencia es la integración dinámica de conocimientos, habilidades, actitudes y valores que desarrollan los seres humanos. En otras palabras, es todo aquello de lo que eres capaz de hacer. Yendo más a fondo, la competencia a la que me estoy refiriendo es específicamente de egreso, la cual tiene también su propia definición, que sería aquella competencia que permite desempeñarte como ciudadano autónomo y flexible dentro de un área profesional.

Partiendo de esto, es verdad que el mantenimiento de software es una parte muy importante del área profesional en lo que se refiere a la administración y producción de proyectos de software, aun así, yo considero que el incluirlo como una competencia es parcialmente correcto, pues basándose en el *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering,* este menciona que al final del curso, el alumno debe tener habilidades que le permitan realizar el correcto mantenimiento del software, mas esta etapa de desarrollo está conformada por mucha partes y no se puede englobar simplemente en la “capacidad de mantener”, sino en las habilidades específicas que le permitan mantener el software, como el uso de métodos y técnicas enfocados a *testear* código complejo y a modificar el código con software de carácter correctivo, adaptativo, preventivo o perfectivo.