Reflexiones unidad 2- Bonilla Pech Russel Adrian

CU vs US

A pesar de tener cosas en común las especificaciones de casos de uso y las historias de usuario no son lo mismo, existen varias maneras de distinguirlas, como el formato, o la extensión que es mucho mayor en la especificación de casos de uso que en las historias de usuario. Básicamente las historias de usuario son descripciones cortas y simples de una característica contada generalmente desde la perspectiva de un usuario quien desea una funcionalidad o capacidad de un software, debido al formato simple que presentan las historias de usuario, además de la facilidad para modificarlas, dichas historias son comúnmente utilizadas en las metodologías ágiles como scrum que asumen que no es posible planificar a muy largo plazo , mientras que las especificaciones de casos de uso describen la manera en la que “un actor” interactúa con un sistema, dichas especificaciones proporcionan detalles, debido a esos detalles tienen una gran extensión y son comunmente utilizadas en las metodologías ágiles las cuales pretenden planificar el proyecto completo.

Mantenimiento del software

Primeramente hay que mencionar que el mantenimiento de software es la modificación de un producto de software después de haber sido entregado a los usuarios o clientes, esto según la norma IEEE 12197, por lo que ya hay una deferencia entre lo que es mantenimiento y lo que es desarrollo de software, sin embargo, esta diferencia no es suficiente, porque en ambos casos se trata de escribir código, entonces tuve que adentrarme un poco más para justificar el hecho de que sí es necesario hacer la diferenciación entre estas dos competencias (mantenimiento y desarrollo de software), bien, después de leer una gran parte de una tesis, identifiqué un punto clave que permite hacer la diferenciación, ese punto clave es el hecho de que “el mantenimiento de software incluye desafíos técnicos y de gestión únicos para los ingenieros de software”, un ejemplo de lo antes mencionado es la habilidad con la que un ingeniero de software debe entender dónde hacer un cambio o una corrección en el software que este individuo no desarrolló, ya que generalmente se trata de software heredado, esta habilidad es sólo una de las diferencias , pero puede significar un justificante y un punto de partida para decir que efectivamente el mantenimiento de software incluye desafíos técnicos y de gestión únicos que podrían estar implicitos en nuestro plan de estudios donde se puede observar que se hace cierta diferenciación entre algunas materias que según la UADY van más enfocadas a desarrollo y otras a mantenimiento de software (ya habíamos hablado de esto en el bloque anterior), por lo antes mencionado, reitero sí es correcto que se haga una diferenciación entre dichas competencias.

Políglotas

Aunque para lograr ser “políglotas” no existe una técnica mágico, hay ciertas características que nos ayudan a lograr la transición más rápida entre lenguajes de programación, la práctica es muy importante, pero la característica básica para lograr una transición más rápida es la lógica de programación, es decir, entender cómo funcionan los lenguajes de programación, ya que existen una serie de patrones que se repiten en todos los lenguajes de programación, como por ejemplo el uso de bucles, condicionales o variables, debido a esto, si entendemos la lógica detrás de los lenguajes de programación y aprendemos dichos patrones, no estaríamos comenzando de nuevo cada vez que queramos aprender, sino que, ya tendríamos una base de conocimiento que nos permitirá avanzar más rápido.

Métodos y técnicas de pruebas

Las pruebas de software son el proceso de ejecución del software para comprobar su funcionamiento, estas pueden ser consideradas desde las etapas iniciales, esto es, desde que se tenga una pieza de software, para apoyar esta idea mencionaré que un método de pruebas del software es el método de pruebas unitarias, básicamente se trata de testear las “piezas más pequeñas del software”, cada pequeño componente, para comprobar el funcionamiento y descubrir defectos. Este método es eficáz, es decir, ayuda a mejorar el desarrollo de software, ya que una vez que se esté seguro que la unidad funciona como debería, los desarrolladores estarían escribiendo sobre un código que está previamente probado.