Reflexiones Unidad 2

Tema 1: Políglotas

En la actualidad existen más de 675 lenguajes de programación, y esto se debe a que cada uno está orientado para realizar una actividad en particular de manera más eficiente que otro, es por ello que puede surgir la duda, ¿cómo puede una persona aprender la mayor cantidad de estos lenguajes en el menor tiempo posible? Antes de reflexionar sobre esta pregunta, vale la pena plantearse otra, ¿es realmente necesario aprender tantos lenguajes de programación? La respuesta breve es no, pero para argumentar un poco esto ejemplifiquemos la situación mediante una analogía más que apropiada: en la vida real, ¿sirve de algo, saber un poco de muchos lenguajes sin conocer alguno en su totalidad? Las personas pueden desarrollarse en su entorno gracias a la capacidad y necesidad innata de comunicarse, es por ello que es necesario establecer lenguajes en todo el mundo, los cuales cambian según las regiones y sociedades, pero ahora supongamos que la siguiente generación de niños solo se les enseñará español hasta los 5 años, y después todas sus clases e interacciones serán en ingles hasta los 10, donde una vez más cambiarán su lenguaje de comunicación principal al japonés, ¿cuál es el resultado? Los chicos de 15 años conocen gran parte de 3 de los idiomas más hablados del mundo, pero son incapaces de comunicarse correctamente entre sí y el mundo que los rodea. Similar a lo anterior, ocurren con intentar dominar todos los lenguajes de programación habidos y por haber si se estima que se necesita entre 6 y 12 meses para comprender uno solo.

Regresando al punto principal, tras la anterior advertencia, si lo que se busca es maximizar el repertorio de lenguajes que una persona conoce, es posible aplicar las mismas estrategias que se suelen usar para la transición de un lenguaje natural a otro:

- Lo primero es conocer en su gran parte un lenguaje de programación, ya que este otorgará las bases sobre lógica de programación, sintaxis y semántica que se pueden esperar en los demás lenguajes. Se recomienda escoger los lenguajes más “robustos”, estáticos y específicos para transitar de manera más fluida hacia los lenguajes más recientes que suelen ser más dinámicos.
- Posteriormente se puede optar por aprender lenguajes de sintaxis similar, esto ya que las estructuras y lógicas resultarán más amenas que si se decidiera aprender un lenguaje con bases totalmente diferentes.
- Finalmente están los lenguajes que distan mucho, en todos los aspectos, del principal que se aprendido.

Aunado a lo anterior, es necesario la constante practica en los lenguajes para mejorar la capacidad de resolución de problemas y mejorar el dominio sobre estas herramientas.

Tema 2: CU vs HU

Tanto los *casos de uso* como las *historias de usuarios* buscan detallar, en menor o mayor medida, las característica o funciones que un sistema debe poseer, pero la diferencia radica en el cómo y el enfoque con que lo hacen, por ello y para diferenciarlas, se describirán las características de cada una de estas herramientas.

- Casos de Uso: Esta es la primera alternativa que se presenta para la determinación de las funciones (o requerimientos) que el sistema a desarrollar debe poseer. Se caracteriza por tener un enfoque orientado a la interacción entre el usuario y el sistema, así como los pasos que realiza cada uno durante este proceso (de ahí su nombre). Está conformado por dos partes o serie de pasos, la primera es realizada por el usuario y la segunda es realizada por el sistema, unas consecuencias de otras. Tienen un aspecto formal e incluso cuentan con un diagrama estandarizado que permite un mejor uso de la herramienta.
- Historias de Usuario: A diferencia de su opositora, esta alternativa es de un carácter menos formal al no contar con un formato estandarizando, esto como consecuencia de partir del lenguaje natural para su redacción. Esta herramienta se caracteriza por tener un enfoque centrado en el usuario más allá de la interacción con el sistema o la descripción del proceso que realiza. Son sentencias de extensión indeterminada que suelen incluir por lo menos tres partes redactadas de la siguiente manera: “como [perfil], [quiero] [para].”
    - El perfil es el usuario objetivo (el destinatario del sistema, puede ser una compañía, un rol, una persona).
    - “Quiero” define lo que el usuario desea hacer, no el *cómo*, por ejemplo, el usuario puede “querer ver películas desde su smartwatch”.
    - Finalmente, “para” explica el por qué este usuario desea realizar dicha acción, qué valor busca obtener al realizar la anterior acción.

Entonces, a forma de síntesis:

- Los Casos de Uso están estandarizados y son de carácter formal, mientras que las Historias de Usuario son informales al no contar con un formato estándar.
- Los CU buscan describir la interacción entre el usuario y el sistema durante la ejecución de una función, mientras que las HU optan por obtener información sobre lo que el usuario espera poder ser capaz de realizar, así como los motivos de ello.
- Es posible afirmar que las HU apuntan más al modelo *user experience*, por esta razón son usados en las metodologías ágiles, por ejemplo, scrum.

Ambas herramientas tienen sus pros y contras, sin embargo, referente a su uso es recomendable emplear CU cuando el proceso se basa en metodologías tradicionales, ¿por qué? Porque no existe gran presencia de procesos iterativos que

permitan refinar la herramienta empleada para obtener y cumplir los requerimientos del cliente, contrario a las metodologías ágiles, donde se puede obtener mejores resultados y sacar el mayor provecho a las HU.

Tema 3: Métodos de Diseño (Interfaz de Usuario)

La interfaz de usuario es una de las partes más importantes en software, especialmente en el de carácter comercial, ya que este representa el vinculo directo entre el sistema y el usuario; a manera de símil, podemos compararlo a la portada de los libros, si no resulta atractiva para el público, estos pasarán de él, es por ello que en la actualidad ha cobrado más importancia en la etapa de diseño de software.

Es entonces cuando surge el concepto de User Interface Design (como parte del User Experience Design) que busca otorgar experiencias satisfactorias a los usuarios de un sistema mediante el diseño de la interfaz de este mediante la búsqueda de una alta *usabilidad*, la cual se puede medir mediante:

- **Capacidad de aprendizaje:** se refiere a la capacidad del sistema para que un usuario pueda aprender a usar sus funciones en un tiempo apropiado.
- **Eficiencia**: una vez el usuario se ha familiarizado con el sistema, este indicador mide que tan fácil, rápido o ameno le es al usuario realizar las acciones que desea.
- **Capacidad de memoria:** este factor busca mesurar la facilidad con que el usuario puede desenvolverse en el sistema tras un periodo de desuso.
- **Prevención de errores:** ¿qué tan expuesto se encuentra el usuario a cometer un error en el sistema por su uso inadecuado? y, ¿cuál sería la consecuencia de que cometa dicho error? Dichas preguntas, entre otras, junto con sus posibles solucione son discutidas en este apartado.
- **Satisfacción:** El usuario ha utilizado el producto, entonces se busca medir su nivel de satisfacción durante el proceso: ¿se sintió frustrado o tranquilo?, ¿cuántas veces ocurrió?, ¿qué no le gustó del producto y que si le gustó?

Un buen trabajo en esta área se ve reflejado directamente en el cliente, y a la larga evita gastos innecesarios y maximiza la recepción y difusión del sistema en cuestión.

Tema 4: Mantenimiento de Software como Competencia

Una de las cuatro principales competencias del programa de estudio de la UADY en el área de LIS es el *mantenimiento de software*, lo cual en primera instancia parece no tener nada fuera de lo habitual, salvo que la existencia de la competencia *desarrollo de software* también forma parte de las cuatro. Por definición, el mantenimiento de software es una de las etapas en el desarrollo de software, entonces, ¿por qué únicamente mantenimiento se considera una competencia específica y las otras etapas no, y, es correcto realizar dicha distinción? Mi respuesta es sí.

La razón por la cual considero que es correcto son las siguiente: comenzando por el hecho de los tiempos de cada etapa, es posible ejemplificar lo anterior con el caso del videojuego Counter-Strike: Global Offensive, dicho programa tomó dos años de desarrollo, comenzando en 2010 y concluyendo con su lanzamiento en 2012, sin embargo, dicho juego sigue recibiendo constante mejoras y actualizaciones a día de hoy, es decir, la fase de requerimiento hasta despliegue tomó dos años, mientras que la etapa de mantenimiento lleva 12 años y contando. Y así como este caso, existen programas con décadas de mantenimiento por detrás que siguen estando vigente a día de hoy. Por esto, es posible afirmar que el mantenimiento puede llegar a ser la etapa más larga del proceso de desarrollo de software.

Aunado a eso, esta etapa tiene un carácter realmente importante en la actualidad: muchas aplicaciones existentes cubren la gran mayoría de necesidades del momento, y aunque no sería correcto afirmar que “ya no existe nada nuevo que crear”, lo cierto es que podría resultar más importante el hecho de mantener vigente los sistemas ya existentes, hacerlos competentes frente a sus similares y cumplir con las nuevas necesidades que pueden surgir en sus clientes.

Cada vez son más los sistemas que se heredan, los servidores que hay que expandir y las aplicaciones que hay que mantener, y es por estas implicaciones que el mantenimiento de software no se puede considerar únicamente una etapa más en el desarrollo de este.