TSP/PSP en Uruguay

Luego de este análisis uno se puede preguntar: ¿Si las inspecciones y revisiones son tan efectivas y tan poco costosas, por qué normalmente no se hacen?

Esta es la primera parte de un conjunto de artículos (seis) que presentan al Personal Software Process (PSP) y al Team Software Process (TSP). También en estos se presentará al equipo TSP Uruguay del Instituto de Computación, UdelaR, a la red "Software and System Process Improvement Network" (SPIN) y al nodo SPIN Uruguay.

Como avanzaremos lentamente por estos temas, y usted puede estar ansioso por conocer el TSP/PSP, qué son los SPIN y qué es el equipo TSP Uruguay, le dejo estos enlaces que rápidamente le darán una idea de cada tema:
- *Breve introducción al PSP:*
http://www.sei.cmu.edu/reports/00tr022.pdf
- *Breve introducción al TSP:*
http://www.sei.cmu.edu/reports/00tr023.pdf
- *SPIN:* http://www.sei.cmu.edu/spin/
- *Grupo de Ingeniería de Software, UdelaR:*
http://www.fing.edu.uy/inco/grupos/gris/

Este primer artículo trata sobre calidad de software, sin entrar en definiciones, estándares y otras cosas que podrían resultar algo aburridas. La calidad es un aspecto central en TSP/PSP y por eso comenzamos este conjunto de artículos de esta manera.

Los tres principios de la gestión de software según Humphrey

Watts Humphrey es, sin lugar a dudas, el gurú de la Calidad de Software. Sus ideas, plasmadas en diversos libros y artículos, fueron la base para el desarrollo de CMM (hoy CMMI). Hablar de los aportes de Humphrey a la Ingeniería de Software llevaría más espacio del que dispongo. Pero, dado nuestro tema, simplemente voy a agregar que es quien desarrolló tanto el PSP como el TSP y que actualmente trabaja en el equipo TSP del Software Engineering Institute (SEI) de la Universidad de Carnegie Mellon.

En el libro "Winning with Software: An executive strategy" (1), Humphrey presenta tres principios fundamentales para la gestión de negocios de software. Dos de ellos son:
1 La calidad es la prioridad más importante.
2 Software de calidad es desarrollado por personas disciplinadas y motivadas.
El tercer principio, que Humphrey lo presenta como el primero, puede parecer extraño a primera vista: "Reconocer que se está en el negocio del software". Hoy en día la mayoría de los negocios dependen fuertemente de una u otra manera del software. Si ese es su caso, entonces usted está en el negocio del software.

Si las demoras en el desarrollo o entrega de software afectan su negocio, si la calidad de algún producto de software también lo afecta, si debido a fallos en el software usted no puede entregar sus productos o servicios, entonces usted está en el negocio del software. En el libro citado anteriormente, Humphrey cuenta que un vicepresidente de Citibank una vez le dijo: "Somos un negocio de software enmascarado como un banco."
¿Reconoció que su negocio es (también) el negocio de software? Si la respuesta es afirmativa entonces debemos considerar los otros dos principios. Una breve discusión sobre el primer principio se presenta a continuación. El segundo principio se presentará en los próximos artículos de esta serie.

Calidad de Software

Para fijar ideas y poder desarrollar un ejemplo, supondremos que el desarrollo de software se realiza en las etapas presentadas en la figura 1. Algunas de estas pueden llamar la atención pero en general representan las etapas más comúnmente encontradas en la industria de desarrollo de software. La figura no presenta ningún proceso de software en particular; estas etapas se pueden encontrar en el desarrollo en cascada, iterativo e incremental, en espiral, por fases, etc.

Figura 1 – Etapas del desarrollo de software

Cada una de estas etapas debe apuntar a construir un producto de calidad. Tanto las etapas de construcción como las etapas de verificación de dicha construcción (revisiones, inspecciones y pruebas).

Lamentablemente, en la industria de desarrollo de software es común que varias de estas etapas no sean realizadas. Las que normalmente se descartan son las revisiones y las inspecciones de todos los productos construidos durante el desarrollo (requerimientos, diseño, código, etc.). Existe una creencia no fundada de que dichas actividades retrasan el proyecto y no aportan a la calidad del producto final. Ninguna de estas dos afirmaciones es cierta.

Fagan en 1976 presentaba las inspecciones como medio para reducir los defectos en el desarrollo de software (2). Luego Humphrey mostraría la efectividad de las revisiones individuales (3). Aplicando estos métodos se obtienen productos de calidad en menor tiempo y en menor costo. Existe una inmensa cantidad de resultados empíricos acerca de la efectividad y el costo de estas técnicas, pero ahora no es el momento de detenernos en esto.

Hace muchos años atrás se hablaba del *code and fix*. Método que hace alusión a "pasar a" programar rápidamente sin realizar diseños y luego "pasar a" las pruebas de sistema sin realizar revisiones, inspecciones ni pruebas unitarias. Codificar, probar y corregir; una y otra vez, parche tras parche. Esta práctica se creyó suprimida a partir de la adopción de procesos de desarrollos de software. Pero esta creencia tampoco es verdad. El *code and fix* sigue tan presente como antes.

Muchos procesos de desarrollo centran la verificación de los productos de bajo nivel (diseño y código) en la realización de pruebas unitarias. Sin embargo, si no se realiza un diseño detallado de bajo nivel y si no se realizan ni inspecciones ni revisiones tanto del diseño como del código, estamos nuevamente presentes ante el famoso *code and fix*. La diferencia con el "antiguo" *code and fix* es que no se espera remover defectos en la prueba de sistemas sino que se intentan remover un poco antes: en la prueba unitaria. Codificar, realizar pruebas unitarias, corregir. *Code and fix*, un poco diferente y moderno, incluso en ocasiones con pruebas unitarias automatizadas, pero de todas formas… *code and fix*.

Menos defectos en el producto final...

Analicemos la calidad durante el desarrollo de software mediante un ejemplo. Para simplificar consideraremos sólo las etapas de codificación en adelante.
Las etapas de verificación (compilación, revisión, inspección y pruebas) finalizan cuando se han removido los defectos que se logran encontrar en esa etapa. Esto quiere decir que un producto que pasa por una de estas etapas queda con menos defectos que con los que ingresó. El porcentaje de defectos que se logran remover en promedio en cada una de estas etapas es conocido como la efectividad de la etapa.

Diversos estudios realizados indican que la efectividad de la compilación y de las pruebas, sean estas unitarias, de integración o de sistemas, ronda el 50%. Esto es lo mismo que decir que en cada una de esas etapas se logra remover la mitad de defectos que contiene el producto. La figura 2 muestra un programa que entra a la etapa de pruebas unitarias con 4 defectos y sale de ella con 2.

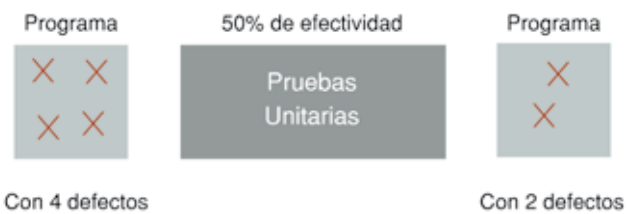

Figura 2 – Efectividad de las pruebas unitarias

También existen diversos estudios acerca de la efectividad de las revisiones y de las inspecciones. Los datos indican que estas técnicas remueven en promedio el 70% de los defectos. Esto es una diferencia sustancial con la compilación y las pruebas.
Estas efectividades se mantienen o mejoran cuando las técnicas son combinadas, es decir, cuando se aplica, una tras otra, distintas técnicas de verificación.

En la actualidad muchas empresas de desarrollo y mantenimiento de software aún no utilizan ni revisiones ni inspecciones. Capers Jones analiza datos de diversas empresas de desarrollo de software y encuentra que el número remanente de defectos ronda los 7,5 defectos cada mil líneas de código (defectos/KLoc) en empresas que no cuentan con ningún nivel del Capability Maturity Model Integration (CMMI) (4).

Agregando al proceso de desarrollo las etapas de revisión y de inspección, las empresas pueden desarrollar productos con aproximadamente 1,05 defectos/KLoc (4). Esta diferencia es sustancial respecto a la práctica más común.

Datos de uso en la industria del TSP indican que los defectos remanentes en los productos rondan los 0,06 defectos/KLoc. La primera vez que conocí estos datos quedé totalmente sorprendido, luego aprendí cómo es que el TSP logra estos números casi mágicos. De este número y otros números hablaremos en la segunda parte de esta serie de artículos. La figura 3 muestra la calidad del producto final según los tres casos mencionados.

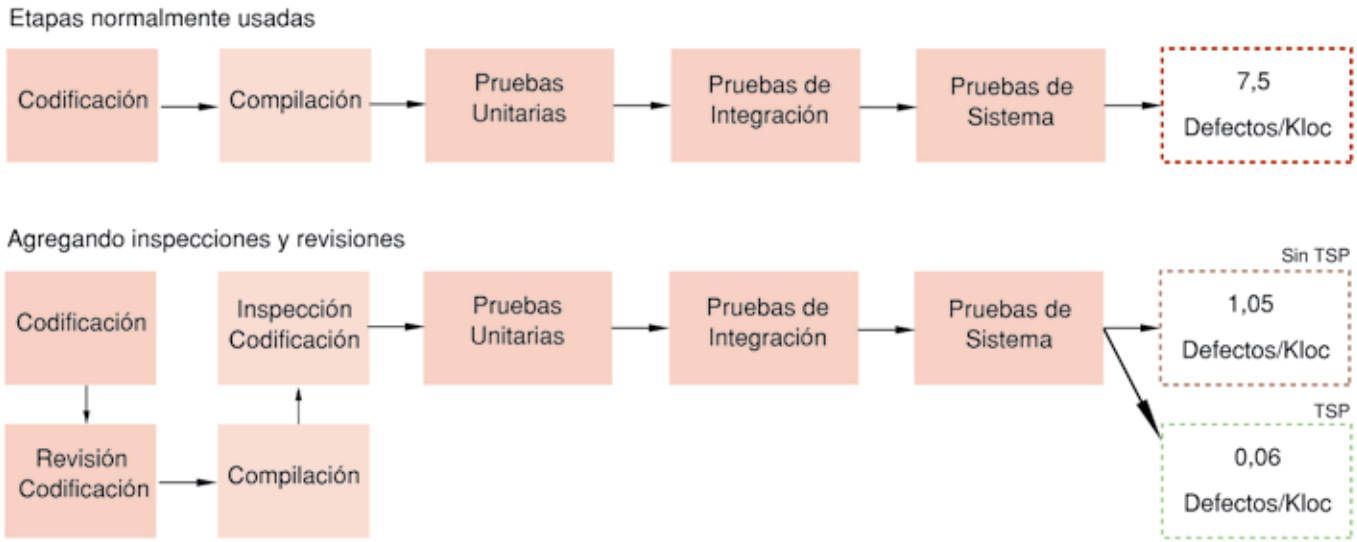

Figura 3 – Defectos remanentes en el producto final

Primera conclusión: realizar inspecciones y revisiones reduce significativamente la cantidad de defectos remanentes en el producto final. Aplicar el TSP reduce en dos órdenes de magnitud la cantidad de defectos remanentes.

... y a menor costo

Aún falta responder acerca del costo. Vamos a definir el costo como la cantidad de horas que le lleva al equipo de desarrollo implementar el producto. La comparación que vamos a realizar es entre la estrategia normalmente usada (que llamaremos estrategia A) y la estrategia con revisiones e inspecciones (que llamaremos estrategia B). Dejaremos TSP fuera de este ejemplo.
El análisis de costos, si bien es sencillo, es un tanto largo y no aporta mucho entrar en detalles. Es importante notar que las diferencias de costos entre ambas estrategias están dadas por dos factores. Uno es el costo asociado a realizar las inspecciones y las revisiones. El otro está asociado al tiempo que lleva remover los defectos en cada etapa.
Como ejemplo usaremos un producto de 100.000 locs. Un producto de esta cantidad de líneas de código termina con 750 defectos si se usa la estrategia A y con 105 defectos si se usa la estrategia B.

El tiempo de corrección de un defecto varía según en qué etapa se encuentre el defecto. Según datos de diversas empresas de desarrollo de software los tiempos promedio de corrección de un defecto rondan los: 3 minutos en revisión, 25 minutos en inspección, 32 minutos en pruebas unitarias y 1.400 minutos en pruebas de sistema.

Simplemente daremos una idea del costo de corregir defectos durante la prueba de sistema en cada una de las estrategias.
La estrategia A remueve 750 defectos durante las pruebas de sistema (50% de efectividad de las pruebas de sistema). Cada defecto se corrige en 1.400 minutos, esto hace un total de 17.500 horas. Una única persona, trabajando sin parar 40 horas semanales las 52 semanas del año tardaría 8,4 años en remover los defectos del producto durante las pruebas de sistema.
La estrategia con inspecciones y revisiones remueve durante las pruebas de sistema 105 defectos. Esto hace un total de 2.450 horas. Una persona tardaría 1,2 años en finalizar las pruebas de sistema. Sumando los costos de las inspecciones y revisiones igual la estrategia B consume extremadamente menos horas que la estrategia A.

Segunda conclusión: en el desarrollo de software la calidad no cuesta, sino que paga. El costo de la calidad se discutirá con mayor profundidad en la tercera entrega de esta serie.

Luego de este análisis uno se puede preguntar: ¿Si las inspecciones y revisiones son tan efectivas y tan poco costosas, por qué normalmente no se hacen? Humphrey indica que esto se debe a que para realizar buenas revisiones e inspecciones se requiere disciplina y entrenamiento especial, y normalmente, los desarrolladores no están entrenados en estos métodos y por lo tanto no creen que estos los ayuden (1). Mark Paulk contesta una pregunta más general sobre PSP de esta manera: "Desafortunadamente, muchas de las mejores prácticas de la ingeniería de software no son tan ampliamente adoptadas como la evidencia recomienda" (5).

Calidad y cronograma

Una preocupación constante de la gerencia, y es bueno que así lo sea, es el cronograma. ¿Podremos entregar el producto a tiempo?

La calidad del producto de software es lo que más afecta el cronograma. El trabajo de calidad se puede predecir, cada etapa tiene un tiempo estimado y se conoce con cuántos defectos remanentes pasa el producto a la siguiente etapa. Esto no sucede así en el desarrollo de mala calidad.

Los proyectos que no gestionan la calidad probablemente no la obtengan, entonces, en esos casos, los proyectos terminarán siendo gestionados por los defectos de los productos desarrollados (1). Cuando los gerentes y los desarrolladores no están entrenados en métodos de calidad los proyectos de desarrollo pasan rápidamente a las pruebas de sistema. Al llegar a estas todo es incierto, se sabe que se está en la etapa de pruebas de sistema pero no se sabe cuándo se va a salir de ella. Los defectos han tomado el control del proyecto y el cronograma probablemente no se cumpla.

En este artículo presentamos de forma breve la importancia de la calidad para un proyecto de software. Proyectos que usan de forma correcta métodos como las revisiones e inspecciones logran obtener productos finales con al menos un orden de magnitud menos de defectos. Además, estos productos se desarrollarán con un menor costo.

Si usted está de acuerdo en que la calidad es la prioridad más importante entonces el segundo paso es saber cómo obtenerla. Los siguientes artículos de esta serie presentarán una forma, probada y medida, de obtener productos de calidad, cumpliendo con el cronograma y los costos establecidos.

Referencias:

1. Humphrey, Watts S. Winning with Software: An Executive Strategy. : Addison-Wesley Professional, 2001.

2. Design and Code Inspections to Reduce Errors in Program Development. Fagan, Michael E. : IBM Systems Journal, 1976, Vol. 15, págs. 182-211.

3. Humphrey, Watts S. A Discipline for Software Engineering. : Addison-Wesley Professional, 1995.

4. Capers, Jones. Software Assessments, Benchmarks, and Best Practices. : Addison-Wesley Professional, 2000.

5. The Impact of Process Discipline on Personal Software Quality and Productivity. Paulk, Mark C. : Software Quality Professional, 2010, Vol. 12, págs. 15-19.

Diego Vallespir
Profesor de Facultad de Ingeniería, UdelaR
TSP Uruguay - Grupo de Ingeniería de Software
dvallesp@fing.edu.uy