Introducción

Mier Lehman y Laszlo Belady formularon a mediados de los años 70 principios de los 80 unas leyes a las cuales se les conocen como las leyes de la evolución del software o leyes de Lehman.

Lehman observó que el gasto estimado que se había realizado en programación en los Estados Unidos en 1977 había sido entre 50 y 100 billones de dólares, lo que suponía un 3% del producto nacional bruto de ese año, y que según pasaban los años iba aumentando.

De ese gasto realizado en 1977 en programación solo el 30% había sido en desarrollo de software, mientras que el 70% restante había sido gastado en el mantenimiento de este, entendiendo mantenimiento como todo cambio que se le hace a un software después de su primera instalación.

Con esto llegaron a la conclusión de que la gran mayoría del software tiene que estar creciendo constantemente para poder adaptarse a los nuevos requisitos. Este crecimiento se ve representado principalmente por la adición de nuevas funcionalidades. Un ejemplo de esto es en el software de código abierto, donde suele estar representado en el lanzamiento de una nueva versión. Cada lanzamiento es el resultado de un incremento del programa, tanto en tamaño como en complejidad.

Al observar que la gran mayoría del software está sujeto a cambios, Lehman y Belady determinaron unas leyes que los cambios obedecerán o deberían obedecer para que el software pueda adaptarse y sobrevivir a lo largo del tiempo.

Clasificación de los programas

Lehman realizó esta clasificación de programas con la intención de hacer comprender la evolución del software y corroborar así sus leyes.

El divide los programas en 3 tipos, que son los siguientes:

- S-Program (static program): se pueden especificar formalmente y derivan de una especificación, es decir, se describe claramente el problema y la solución a la que se quiere llegar. Por ejemplo, el mínimo común múltiplo de dos enteros, o el problema de las ocho reinas. Son problemas que se resuelven de una manera, y esta está definida por su especificación, de tal forma que guía al programador a conseguir su único objetivo, alcanzar la solución deseada. Cuando un programa es aprobado legítimamente el programador se puede preocupar por aspectos como la eficiencia y la elegancia pero siempre sin alterar el mapeo entrada-salida que se especifica en el problema. Si se altera algo de esto a interés del usuario estamos dando lugar a una solución para un nuevo problema.
- P-Program (practical program): en este caso la declaración del problema es una abstracción del mundo real, no existe una manera exacta de solucionarlo o si existe, no se puede llevar a la práctica de una manera eficiente. Por lo tanto, lo que se va a conseguir es una aproximación a la solución del mundo real, y para llegar a ella nos encontraremos con incertidumbres, incóginitas y demás factores. Por ejemplo, la predicción meteorológica o el ajedrez. Aquí el objetivo es llegar a una solución que sea válida y aporte valor al contexto del mundo real. El problema es específico, pero el llegar a la solución depende de un proceso iterativo y cambiante, por lo que estos tipos de programas están sometidos al cambio.
- E-Program (embedded program): Aun más propensos al cambio. Programas que mecanizan una actividad humana o social. La diferencia de esta estructura de programa con el resto es que aquí el programa forma parte del mundo que modela, está incrustado en él. Por

ejemplo, el control de tráfico aéreo o los sistemas operativos de ordenadores. Son problemas cuya solución surge del estudio y análisis de varias personas, es decir, se necesitan varios puntos de vista. Por lo tanto, la implementación de la misma depende de los desarrolladores, lo que hace que a la hora de lanzar la aplicación surjan tantas dudas sobre su validez. Es aquí cuando se tendrán que realizar comparaciones de la salida del programa con la aplicación del mundo real. Una vez que los desarrolladores dominen su propio diseño se esforzarán por mejorarlo de tal manera que faciliten el cambio futuro y mejore la eficiencia del programa. Otros aspectos de cambio pueden ser avances tencológicos, nuevo hardware...

La mayoría de los programas hoy día son propensos al cambio, perteneciendo a los programas de tipo P y E, que en su conjunto forman el grupo A. Por ello Lehman creo sus leyes citadas en el siguiente apartado para describir que el software siempre evoluciona al ser dependientes de su contexto.

1 Ley de Lehman. Cambio continuo

« Los sistemas deben ser continuamente adaptados o se convierten progresivamente en menos satisfactorios ».

Tarde o temprano será necesaria una adaptación al cambio. La adaptación no implica necesariamente un crecimiento del sistema, ya que un proceso de adaptación puede estar basado en una nueva forma de enfocar las funcionalidades ya implementadas.

Es cierto que hay sistemas que no se han tocado en años, pero el motivo principal es económico, a lo que hay que sumar el miedo de retocar un sistema que está funcionando.

Nuestra inversión no termina con la adquisición del producto software o con su desarrollo a medida, sino que tendremos que seguir realizando nuevas inversiones para adaptar el mismo a los cambios en las necesidades de la organización y el mercado.

2 Ley de Lehman. Complejidad incremental

« Cuando un sistema evoluciona se incrementa su complejidad a menos que se trabaje para mantenerla o reducirla ».

La complejidad crece no solo a nivel de deuda técnica, sino también a nivel de administración, usabilidad y recursos software y hardware necesarios para su funcionamiento.

Si se aplican buenas prácticas, este incremento de la complejidad puede hacerse más moderado e incluso puede reducirse si se aplican medidas específicamente destinadas a este fin.

Con el paso del tiempo va a ser necesario evolucionar las aplicaciones y ese coste será progresivamente mayor conforme se realicen nuevas actividades de mantenimiento en el sistema.

3 Ley de Lehman. Autoregulación

« El proceso de evolución de un sistema es autorregulado con una distribución de las medidas del producto y del proceso cercana a la normal ».

La complejidad del software interviene como un elemento más de el proceso autorregulador, de esta forma podemos asociar esta ley con la segunda:

– Podemos realizar mejoras, limitadas por las propias características del sistema y su contexto, es decir, se puede mejorar, pero siempre habrá una «carga extra» que tendremos que soportar.

– En el caso de que evolucionemos el sistema, habrá una serie de atributos que permanecerán constantes, como podría ser la tasa de errores, o que crecerán proporcionalmente como el tamaño del sistema.

4 Ley de Lehman. Conservación de la estabilidad organizacional

« La velocidad (y efectividad) de desarrollo de un sistema en evolución permanece invariante durante su ciclo de vida ».

Es complicado que se den todas las condiciones adecuadas para que la velocidad de desarrollo de un determinado sistema aumente, ya que cuando se tenga un buen equipo de trabajo, los responsables funcionales o el personal directivo puedes mostrarse indecisos, toman decisiones erróneas, tienen que reducir o cambiar el equipo por motivos económicos u optan por ralentizar la evolución o cuando estas circunstancias sean favorables.

A lo anterior hay que sumar otros factores como son el incremento de la complejidad que compensaría con un mejor funcionamiento de la organización o el hecho de que haya algunos atributos que según la tercera ley de Lehman se mantienen constantes como, por ejemplo, la tasa de errores.

Haciendo medias de esas situaciones se entenderá que la efectividad de la evolución se mantiene más o menos constante durante su desarrollo.

5 Ley de Lehman. Conservación de la familiaridad

«Cuando un sistema evoluciona, todos aquellos que están asociados a él: desarrolladores, personal de ventas, usuarios, deben mantener un conocimiento de su contenido y comportamiento para tratar de conseguir que la evolución sea satisfactoria. Un crecimiento excesivo disminuye ese conocimiento. De ahí que el crecimiento incremental promedio permanezca invariante».

El sistema debe evolucionar de manera sostenida para que el grupo de personas vinculadas a él no pierda familiaridad y, en consecuencia, eficiencia en el desarrollo.

6 Ley de Lehman. Crecimiento continuo

«Las funcionalidades del sistema tienen que crecer constantemente para mantener la satisfacción del usuario a lo largo de su ciclo de vida».

Esta ley se basa en dos fundamentos:

- Durante el proceso de desarrollo se suelen descartar funciones por falta de tiempo, presupuesto... o bien porque no se consideraban esenciales en ese momento.

- En ocasiones los desarrolladores pretenden ocultar deficiencias en funcionalidades añadiendo otras nuevas.

Hoy en día esta ley se tiene muy en cuenta y los desarrolladores intentan encontrar la solución más simple que satisfaga las necesidades del usuario y solo añadir funcionalidad en caso de que sea estrictamente necesario

7 Ley de Lehman. Reducción de la calidad

«La calidad de los sistemas comienza a disminuir a menos que se mantengan de forma rigurosa y se adapten a los cambios en su entorno de funcionamiento (operacional)».

Esta ley se enfoca desde tres puntos de vista

El usuario es mñás exigente conforme más tiempo invierte en el sistema

La calidad del software se ve a afectada por su crecimiento. Aspectos como la usabilidad, seguridad o la deuda técnica van siendo más díficiles de llevar conforme crece el proyecto.

Es importante mantener actualizadas todas las características que utilice el sistema.

8 Ley de Lehman. Realimentación del sistema

«El proceso de evolución del sistema es consecuencia de un proceso de realimentación (feedback) a diferentes niveles, de manera iterativa y por diferentes actores, y debe ser considerado como tal para conseguir mejoras significativas sobre cualquier base razonable».

En cada evolución del sistema existe la posibilidad de recabar feedback a diferentes niveles con el que podemos generar nuevas actividades que nos permitan generar un sistema más correcto y completo