Algunas recomendaciones a aplicar en el diseño de interfaces de software

*Gerardo Cerda Neumann**
*Rodolfo Guzmán Zúñiga***

Resumen

Si bien la ingeniería de software ha mejorado la calidad y confiabilidad del software, existe un área que ha sido descuidada hasta el momento: el diseño y construcción de la interfaz del usuario. Gran cantidad del esfuerzo en el desarrollo del software está invertido en el diseño y construcción de la base de datos y del código necesario para manipularla, y de las comunicaciones pertinentes para que estos datos se puedan accesar. Sin embargo, lo único que tiene a la vista finalmente el usuario es la interfaz. Si esta es difícil de utilizar, no importará toda la calidad que posea el resto del software, el usuario se frustrará y se sentirá incómodo cuando tenga que interactuar con el programa. Poco esfuerzo se dedica a crear un diseño de calidad de dicha interfaz, que tome en cuenta las características particulares tanto del usuario como del entorno en que será usado el software.

En un artículo anterior (Akadèmeia, Vol. 2, N° 2, diciembre de 2002) se presentó y justificó la importancia de construir interfaces humano-computador de alta calidad. En este artículo se continúa con el tema, presentando algunas "leyes" útiles a aplicar al momento de diseñar dicha interfaz, además de una técnica útil para evaluar aquellas interfaces antes de construirlas.

* Ingeniero Informático, Pontificia Universidad Católica de Valparaíso; magíster en Ingeniería Informática, Usach. Académico, Escuela de Ingeniería, UCINF.
** Alumno de Ingeniería Civil en Informática, UCINF.

1. Introducción

En todos los métodos de desarrollo de software se incluye la actividad de diseño de interfaz (Cerda, 2003). Al fin y al cabo, la interfaz será lo único que el usuario tendrá a la vista mientras trabaja con el software. Parece natural, entonces, que dicha actividad sea de gran importancia y que se le dediquen muchos recursos y esfuerzos. Sin embargo, normalmente no ocurre así, ya que es poco el tiempo destinado y además su ejecución se le asigna a personas con gran experiencia en la programación, pero con escasos conocimientos tanto en elementos gráficos como en la interacción. Así, muchas veces el resultado redunda en que la interfaz desarrollada está llena de ventanas de diálogo (que piden confirmar una y otra vez las acciones que se quieren ejecutar), de extraños mensajes de error, de opciones usadas frecuentemente ocultas en submenús y de opciones usadas rara vez que, paradójicamente, están siempre a la vista. La lista de errores de interfaz encontrados en cualquier programa computacional podría ser mucho más extensa aún. Diseñar y construir una buena interfaz de usuario puede generar grandes ahorros a la organización que la utilice. Esto ha sido ratificado en varios estudios, entre ellos, en uno realizado por la Carnegie Mellon University, que determina el ahorro en US $6.800.000 para una aplicación compleja utilizada por 240 mil empleados (citado en Sanz et al., 1996).

No obstante, el objetivo de este trabajo no se reduce a dar a conocer esta negativa realidad sino además intenta proponer una serie de recomendaciones (presentadas como "leyes") que ayuden al diseño y construcción de interfaces. Por esta razón se irán presentando cada una de ellas a través de ejemplos que permitan comprender mejor su utilidad. Finalmente, se presentará un sencillo método para evaluar la calidad de una interfaz humano-computador, que evite una apreciación centrada en lo subjetivo y personal.

Cabe destacar que se ha mantenido una parte de la información contenida en el artículo referido de Cerda (2002) para darle más estructura a lo planteado en este nuevo texto.

2. Recomendaciones para lograr una mejor interfaz humano-computador

A continuación se presentan algunas de las "leyes" que los autores han descubierto consultando diferentes fuentes bibliográficas y en los proyectos académicos en los que les ha tocado dirigir y participar.

Ley 1: Mantener sencillo lo sencillo: consiste en que lo que es fácil de usar, lo siga siendo y no se complique de manera artificial.

A modo de ejemplo se cita el ajuste de un reloj de videograbadora. ¿Quién no ha visto la hora "pestañeando" en uno de estos equipos? Se podrá argumentar que ello se debe a cortes de electricidad; sin embargo, lo más probable es que el dueño no quiera poner la hora correcta debido a lo complicado que le resulta el ajuste.

En este sentido se propone la interfaz (Raskin, 2001) presentada en la figura 1:

¿Para qué complicar más las cosas?

Es cierto que las tareas complejas pueden requerir de interfaces complejas, pero no hay motivo para complicar las tareas que son sencillas.

Ley 2: Retroalimentar en forma conveniente a los usuarios: se parte del supuesto de que los usuarios necesariamente se equivocarán (incluso los más expertos). Por este motivo, los mensajes del software deben ser lo más claros posibles (debe evitarse la costumbre de poner la palabra ERROR, puesto que induce al usuario a sentirse como estúpido).

Una buena analogía es posible de encontrar en la experiencia de manejar

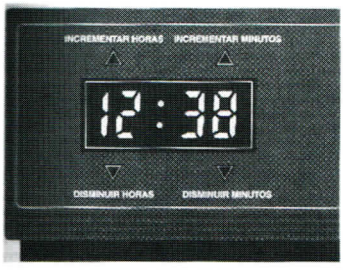

Figura 1: Ejemplo de interfaz fácil de usar.

La forma más simple es contar con cuatro botones: uno para aumentar las horas, otro para sumar minutos, uno para disminuir las horas y finalmente otro para restar minutos.

un automóvil. El conductor acelera, frena y mueve el volante, sintiendo de manera instantánea que el automóvil responde. Como el software puede entregar sensaciones físicas aun de manera muy sencilla (y utilizando hardware complejo y caro), lo mejor es que el usuario tenga frente a sí claros indicadores de lo que está sucediendo (Sommerville, 2002). En este sentido, los despliegues de información dinámica (que han sido creados imitando indicadores usados hace mucho tiempo por las personas) resultan muy útiles, tal como se muestra en la figura 2:

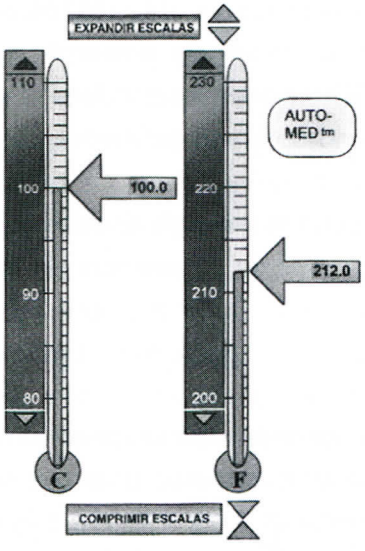

Figura 2: Ejemplo de interfaz que retroalimenta en forma conveniente al usuario.

En esta interfaz, el usuario va cambiando una de las escalas (Celsius o Fahrenheit) y en forma instantánea cambia el otro termómetro.

Ley 3: Mantener un estilo definido: cuando un software ha sido diseñado por distintas personas, el usuario se da fácilmente cuenta de ello. Para la misma situación aparecen mensajes diferentes o, a la inversa, para situaciones distintas se utiliza el mismo mensaje. En este sentido, resulta muy importante mantener un estilo gráfico consistente. Esto implica, por una parte, usar tipos de letras, fondos y colores de manera coherente (siempre igual) y, por otra, lograr una combinación armoniosa.

A modo de ejemplo se muestran dos interfaces del mismo software en la figura 3. La presencia del mismo estilo en ambas es evidente, lo que contribuye a la tranquilidad del usuario ya que siente que está frente a algo "conocido" (Kristof, 1998).

Figura 3: Ejemplo de interfaces que conservan un estilo.

En ambas interfaces se mantienen los mismos iconos y fondos, lo que permite una fácil transición de una a otra.

Esta ley, entonces, se puede resumir así: se hace necesario mantener elementos gráficos conocidos y coherentes para el usuario.

Ley 4: El mínimo esfuerzo: el usuario de la interfaz sólo debe digitar el mínimo indispensable.

A modo de ejemplo se puede citar:

- La fecha del registro debe ser propuesta por el software y el usuario sólo la confirma. En caso de que sea necesario, la cambia por otra digitándola.
- La opción de impresión debiera tener en forma predefinida el papel y la calidad de impresión que se utilizan con mayor frecuencia.
- Al momento de ingresar el Rut de una persona el sistema deberá calcular el dígito verificador en forma automática.

Así, esta ley se puede resumir de la siguiente manera: el usuario debe hacer la menor cantidad de digitaciones y clicks de menú.

Ley 5: Unificación de mensajes y textos: antes de empezar a construir el programa se debe lograr un consenso respecto a los títulos de las ventanas, los *captions* de los botones, las ayudas (*hints*), y en cuanto a cómo resaltar las opciones que están activas (por ejemplo, mostrar claramente cuál es la "pestaña" que se está usando). Con respecto a los mensajes, botones y títulos de las ventanas, lo ideal es que se usen tecnicismos acorde con el ambiente en el cual se va a insertar el sistema, en lo posible, términos relacionados con el rubro de la organización.

Se debe generar una lista con todos los mensajes y títulos de la interfaz. Esta lista debe ser validada tanto por el cliente como por el usuario.

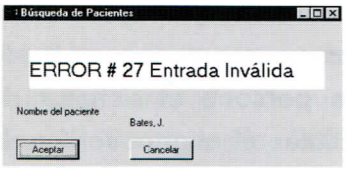

Figura 4: Ejemplo de un mal mensaje para el usuario.

Evidentemente, mensajes como este no orientan para nada al usuario y, por el contrario, lo ponen nervioso y molesto.

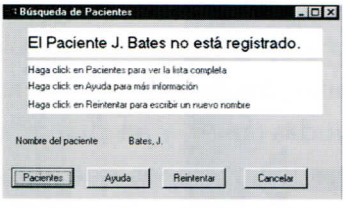

Figura 5: Ejemplo de un buen mensaje para el usuario.

Como contraparte, mensajes como este ayudan al usuario a entender cómo utilizar el software en su propio provecho.

Ley 6: Tener un rol de "diseñador de interfaz": cuando todo el mundo es responsable de la calidad de la interfaz, en realidad nadie toma esta responsabilidad en sus manos (Cooper, 2001). Es demasiado fácil asumir que otra persona en el equipo de desarrollo solucionará cualquier dificultad de interfaz que surja. Además es importante recordar que los programadores tenderán siempre a codificar una interfaz que a ellos les sea más cómoda de usar o más fácil de programar.

El diseñador de la Interfaz es, por tanto, la persona que se encargará de proponer una interfaz que resulte cómoda para el usuario, a la vez que posible de construir por parte del equipo de desarrollo. Por ende, quien asuma esta responsabilidad debe poseer conocimientos tanto de diseño y evaluación de interfaz como de programación. Asimismo, debe contar con la confianza de los programadores, ya que ellos tendrán siempre la tendencia a codificar lo que más les acomode, sobre todo en cuanto a tiempo de desarrollo. El diseñador de interfaz es la persona encargada de construir los guiones, que luego debe presentar al equipo de desarrollo. Además, debe ser capaz de

evaluar las interfaces propuestas, al menos, usando el modelo GOMS que se presentará más adelante.

Una vez presentados todos los conceptos en que se basa el método propuesto es posible explicitar sus diferentes fases y objetivos.

Ley 7: Accesibilidad que cumpla con "todo a la vista": este principio se refiere a que la interfaz tenga todo lo que el usuario desee, nada más, pero tampoco nada menos. El ideal es que se brinde toda la funcionalidad "a un click de distancia". A fin de cuentas se trata de entregarle al usuario la cantidad justa de pasos a seguir para sacar provecho de la interfaz, generando así un mínimo porcentaje de error y un alto porcentaje de rendimiento y satisfacción. Ahora bien, este término es más conocido como *Look and Feel*, concepto orientado a la programación de los entornos gráficos, como JAVA.

Ley 8: Personalización que cumpla con "no tocar": se refiere a que el programa no cambie las modificaciones realizadas por el usuario, es decir, que no toque la interfaz personalizada (Sánchez, 2002). Con esto se avanza un paso más en entregar lo que el usuario desea, esto es: trabajar en un entorno grato y conocido. Entonces, ¿qué mejor si es el usuario quien define cómo verá sus aplicaciones? Obviamente, no podrá tener acceso a las partes preestablecidas o inalterables. Por otra parte, un ejemplo específico es el caso de Yahoo, que en su servicio de correo dio el primer paso en las personalizaciones, ya que permite en la sección MyYahoo!, crear un entorno gráfico tal como el usuario lo desee, sincronizando los datos con una Palm, ver grupos de noticias, perfiles dentro del los juegos online, entre muchas cosas más.

Ley 9: "Usabilidad" que cumpla con la "intuitividad": usabilidad es una mezcla entre intuición y facilidad, que permite al producto o servicio impactar e influenciar directamente sobre la productividad, rentabilidad y eficiencia de un negocio, *on* y *off line* y, con ello, la experiencia de usuario en general (Sanz et al., 1996).

Las estadísticas demuestran que la inversión en la mejora de la usabilidad y la experiencia de usuario es muy rentable para la empresa. En un

estudio se sitúa el rango costo/beneficio de la usabilidad en 1$/10$-100$. Es decir, por cada dólar invertido en usabilidad, la empresa obtiene un beneficio de entre 10 y 100 dólares, ya sea por mejora de la productividad o por la reducción de costos. Una inversión orientada a mejorar la usabilidad y la experiencia de usuario en webs corporativos, intranets, extranets, sitios de *E-commerce* y entornos *wireless* se traduce en importantes beneficios para la empresa y, al mismo tiempo, aumenta el grado de satisfacción del cliente. La inversión en usabilidad siempre resulta rentable y la mejora en la usabilidad impacta en forma directa la experiencia del usuario. Con dicha mejora logramos, además, obtener un incremento en el volumen de gasto por usuario y por compra (razonable), aumentando potencialmente las ganancias.

3. Evaluación de interfaces

Todo lo mencionado anteriormente sin duda apoyará a los diseñadores de interfaces humano-computador que quieran construir mejores productos; sin embargo, esto no estaría completo si no se dispusiera de alguna manera de evaluar la interfaz antes de construirla. Para esto se propone utilizar el método GOMS (acrónimo de los conceptos Goal, Object, Method y Selection).

El modelo GOMS fue presentado por primera vez en 1983 en una publicación de los autores Stuart K. Card, Thomas P. Moran y Allen Newell titulada *The Psychology of Human-Computer Interaction*. Se basa en la cuantificación de los tiempos utilizados al interactuar con los dispositivos de interfaz del computador (Raskin, 2001).

Se utiliza la siguiente nomenclatura:

K = 0.2 segundo

Tecleo: tiempo que toma oprimir una tecla del teclado.

P = 1.1 segundo

Señalamiento: tiempo que requiere un usuario para señalar una posición en la pantalla.

H = 0.4 segundo

Inicio: tiempo que toma a la mano de un usuario moverse del teclado al mouse o viceversa.

R =

Respuesta: tiempo que debe esperar un usuario para que un computador responda a una entrada.

A modo de ejemplo, se presenta la utilización de la siguiente interfaz en la figura 6, donde un usuario desea saber si un pariente suyo está internado en el hospital, para lo cual digita el nombre de dicha persona.

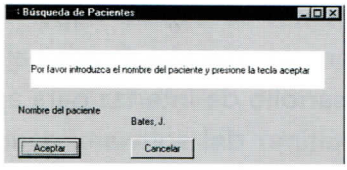

Figura 6: Interfaz de ingreso de datos.

En este caso el usuario debe:

1) Poner la mano sobre el mouse.
2) Mover el mouse hacia el área donde va a digitar.
3) Mover la mano desde el mouse hasta el teclado.
4) Escribir los ocho caracteres del nombre del paciente que desea buscar.
5) Mover la mano desde el teclado hasta el mouse.
6) Mover el mouse sobre el botón de "Aceptar".
7) Hacer click sobre el botón.
8) Esperar la respuesta de la búsqueda.

Aplicando GOMS los tiempos son:

1) H
2) P
3) H
4) KKKKKKKK
5) H
6) P
7) K
8) R

Aplicando este sencillo modelo se puede concluir que el tiempo requerido para esta acción precisa de: 0.4 + 1.1 + 0.4 + (8 * 0.2) + 0.4 + 1.1 + 0.2 + 3 = 8.2 segundos. De este modo es posible comparar esta interfaz con otra y concluir cuál necesita ocupar menos tiempo para encontrar a un paciente. Por ejemplo, se podría definir una interfaz que sólo esperara que el usuario escriba directamente el nombre del paciente buscado y luego presione la tecla <Enter>. Esta interfaz requeriría sólo de: KKKKKKKK + K + R; es decir, bastaría con presionar las ocho teclas del nombre y luego la tecla <Enter>, lo que da un tiempo de (9 * 0.2) + 3, vale decir, 2.1, lo que en definitiva es más óptimo.

4. Conclusiones

Es evidente que para aplicar las leyes aquí propuestas se asignarán muchos más recursos que los entregados hasta ahora al desarrollo de la interfaz. Esto provocará bastante sorpresa en los equipos de desarrollo de software más acostumbrados a asignar recursos a la creación de bases de datos o de aplicaciones de comunicaciones que a pensar en la interfaz. Sin embargo, al revisar el sustento conceptual de lo planteado se llega a la conclusión de que no es posible seguir construyendo software sin dedicarle más tiempo a la interfaz, so pena de que el usuario simplemente no lo utilice o lo haga sin ánimo.

Las leyes planteadas pueden ser incorporadas a cualquier método de desarrollo de software, condición que las hace muy prácticas y aplicables.

Finalmente, se espera haber hecho una contribución significativa al tema de desarrollo de interfaz para apoyar el fin último del software: ayudar al usuario en el logro de su objetivo de una manera sencilla y cómoda.

5. Bibliografía

Card, Stuart K.; Thomas P. Moran and Allan Newell. *The Psychology of Human-Computer Interaction*. Mahwah, N.J.: Laurence Erlbaun Associates Inc., 2000.

Cerda Neumann, Gerardo. "¿Estamos diseñando las interfaces de software correctas?" *Akadèmeia* Vol. 2 N° 2, (2002): 5-15.

—. "Propuesta de método para desarrollar software que posea una interfaz multimedial". Trabajo aceptado en el evento "Informática 2003". La Habana, Cuba.

Cooper, Alan. *Presos de la tecnología*. México D.F.: Prentice Hall, 2001.

Kristof, Ray y Amy Satran. *Diseño interactivo*. Madrid: Anaya Multimedia, 1998.

Raskin, Jef. *Diseño de sistemas interactivos*. México D.F.: Addison Wesley, 2001.

Sánchez Alcocer, Felipe. "MyUIML Editor: Editor de componentes personalizados para interfaces de aplicaciones vía Internet". Artículo extraído de http://www.pue.udlap.mx/~tesis/lis/sanchez_a_f/ (sitio web, Escuela de Ingeniería de la Universidad de las Américas, Puebla, 2002).

Sanz, Marcos F.; Enrique J. Gómez y Francisco del Pozo. "Coste y beneficio de la ingeniería de usabilidad". Artículo extraído de http://www.tid.es/presencia/boletin/bolet10/art007.htm. *Boletín Factores Humanos* 10 (abr., 1996).

Sommerville, Ian. *Ingeniería de Software*. 6ª ed. México D.F.: Addison Wesley, 2002.