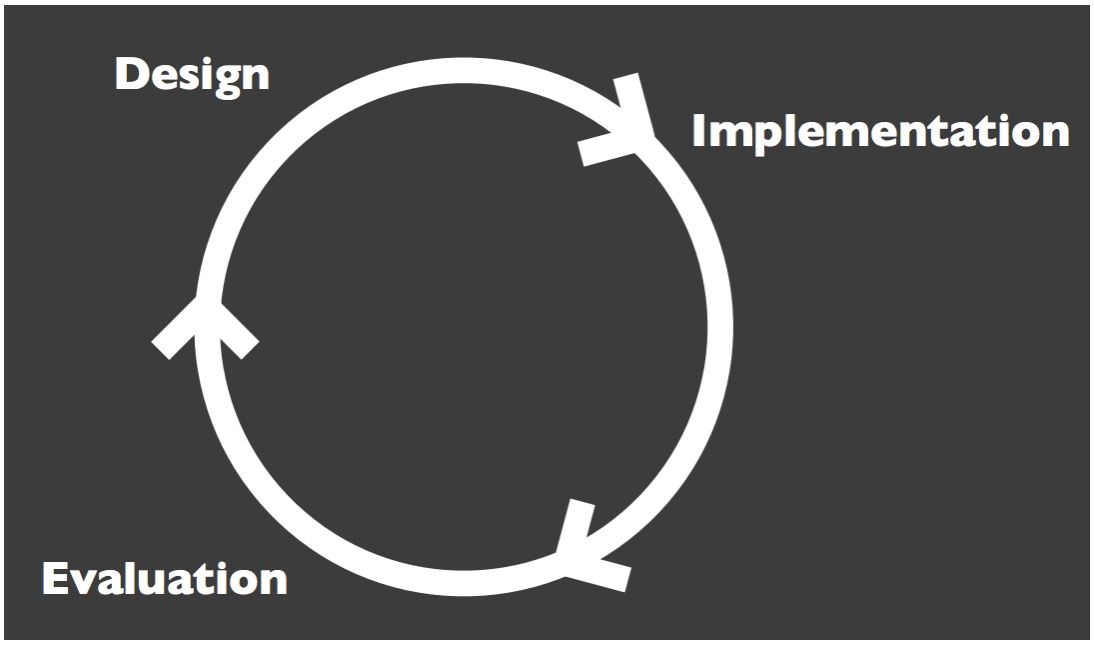
Human Computer Interaction

Nicolás Hock Isaza - 200727001010

El curso de interacción humano-computador (HCI) inicia con una breve historia de "qué es HCI" y se da una visión muy amplia de como se aplica. Pero, básicamente se dice que es un ciclo entre ***diseño***, ***implementación*** *y* ***evaluación*** de la interfaz del usuario para usar el computador.



Un buen diseño hace que las personas disfruten el uso de las cosas. Mejora la experiencia del usuario con ellas y ayuda a que las personas realicen las tareas que necesitan de una manera más fácil y cómoda. Una buena interfaz tiene un efecto gigante en la capacidad que tienen las personas de realizar estas funciones. Puede mejorar la eficacia de las personas y volverlas más productivas.

Un mal diseño es muy costoso. Se pierde tanto dinero como tiempo y en casos extremos se pueden perder vidas. Por ejemplo, un aparato médico que sea difícil

de utilizar o que no muestre la información necesaria en el momento adecuado puede ser mortal.

Muchas veces arreglar estos problemas de las interfaces consiste en seguir ciertas reglas y principios "comunes". Cosas como la consistencia entre las interfaces, donde los colores signifiquen lo mismo en cualquier pantalla del software, pueden ser una gran ayuda y una razón menos para que el usuario se preocupe.

A la hora de crear soluciones (software o no), es muy importante tener siempre presente que el *fin* de la herramienta es solucionar algo. No aprender a usar la herramienta. **El fin de la herramienta va más allá de usarla**.

Sin llevarlo a extremos de salvar vidas, pensemos que si una herramienta nos demora 10 minutos (en promedio) para usarla (como la página de un banco, donde es imposible encontrar lo que se desea hacer), se tienen que se pierden, en Colombia (con 44 millones de habitantes), casi 7.5 millones de "horas hombre" por esos 10 minutos. Eso es mucho tiempo desperdiciado en *nada*.

Lo que se busca realmente es que la interfaz sea transparente. Esto se puede obtener por diseño, práctica (del usuario con la herramienta) o una combinación de los dos. Pero el fin de una buena interfaz es, precisamente, *no existir* bajo los ojos del usuario. Siempre que la herramienta funciona como el usuario *espera* que funcione, podemos hablar de una buena herramienta. Así la concentración del usuario pasa de *cómo* usar la herramienta a *usar* la herramienta. Claramente un buen diseño hace que la experiencia del usuario con la herramienta, para saberla usar, disminuye.

Prototipos

El concepto más importante del que se habla en el curso es el de un ***prototipo***. Dado que el desarrollo es un ciclo, el prototipo entra como una pieza fundamental, sobre todo iniciando, que permite obtener retroalimentación de usuarios y personas reales.

Un prototipo sirve para probar una idea de diseño específica, puede ser el tamaño del dispositivo o la experiencia en general con alguna herramienta. Un prototipo **no** debe ser avanzado ni costoso y **siempre** se debe pensar que será un prototipo y no un producto final, es decir, siempre tiene el derecho de *retirarse*. El objetivo de un prototipo no es el artefacto como tal, es la *retroalimentación* que genera.

Un prototipo es una forma de tener una visión común entre varios *stakeholders* sobre el diseño o el producto que se está implementando. Muchas veces, por más documentos y explicaciones que existan, dos personas tendrán una visión diferente de las cosas en sus mentes. Un prototipo ayuda a *homogenizar* estas visiones y tener un punto en común en el avance y desarrollo que se ha hecho y hacia donde se quiere ir.

Hay muchos ejemplos interesantes acerca de prototipos, por ejemplo, en los 90, Kodak contrató la empresa de diseño IDEO para BLA BLA BLA una cámara donde se pudieran revisar las fotos después de tomarlas, desde la cámara. El prototipo (abajo) consistía en solo el casco de la cámara. Ningún elemento fotográfico se puso en el prototipo, pues la idea era conseguir retroalimentación con la experiencia del usuario en las nuevas funciones de la cámara y no en la parte fotográfica. La cámara que salió de este proyecto es la Kodak DC210.

Claramente el prototipo es mucho más grande que el producto final, pero la idea no era investigar como hacer los componentes pequeños, era identificar las interacciones que los usuarios esperaban al usar una cámara digital donde se podían editar fotos. De hecho, todos los cálculos y computaciones de la edición de imágenes se hacían en un computador conectado al prototipo.

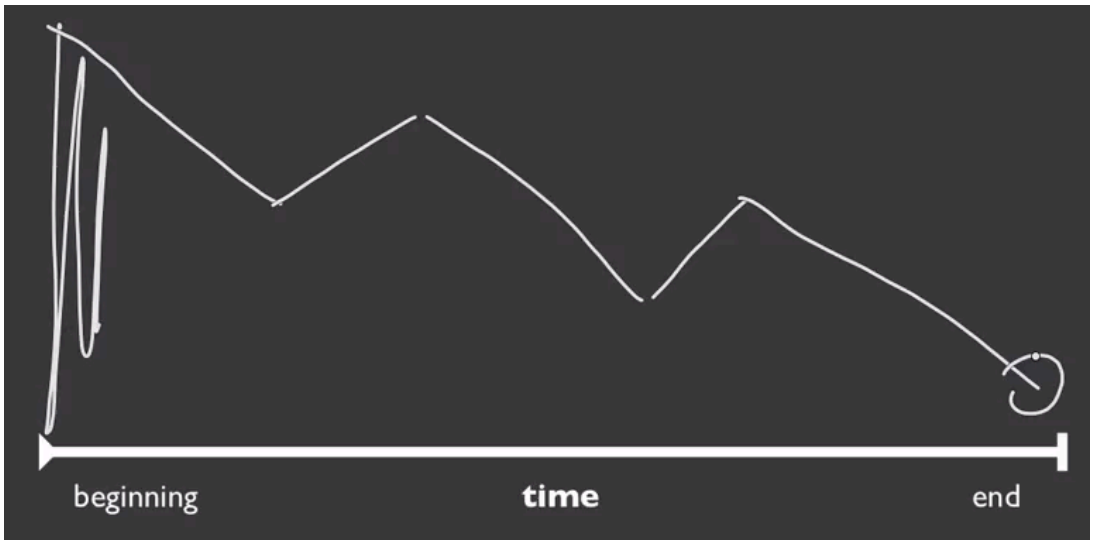
Realizar un prototipo es una estrategia para tratar con cosas que son difíciles de predecir.

Los prototipos permiten a los desarrolladores del producto concentrarse en el **objetivo** del producto, mientras **evolucionan** el diseño con retroalimentación rápida y real.

El proceso de un prototipo

Aún cuándo un prototipo es una gran estrategia para recibir retroalimentación de los usuarios y la interacción con las idas que se presentan, las etapas de un prototipo varían mientras se avanza en el proceso de diseño. El claro que mientras más adelante se esté en el proceso, más información se debe tener. Esto incluye haber eliminado ciertas ideas consideradas previamente.

Se tiene que en un principio se deben explorar una gran cantidad de ideas. Siempre buscando agilidad y facilidad para hacer prototipos nuevos, que se puedan desechar sin problemas. Una vez se haya aprendido algo se buscará mejorar los mejores diseños, se enfoca el desarrollo en estos y acá se pueden probar varias ideas nuevas, pero no tantas como antes… así en cada etapa del desarrollo. Al final los cambios que se hacen van siendo mucho menores (como colores o tipo de letra) y el prototipo más "fiel" al desarrollo final.



Pero, ¿qué nos enseña un prototipo?

- **Implementación:** ¿Cómo puede funcionar la herramienta?
- **Visual**: ¿Cómo se podría *ver* la herramienta?
- **Rol**: ¿Cómo sería la *experiencia* con la herramienta?

Es importante resaltar que un prototipo no es necesariamente algo pequeño. Cuándo Boeing estaba diseñando los primeros aviones para vuelos entre costas de E.E.U.U, se hicieron prototipos donde llevaban personas y las sentaban por seis o siete horas para simular toda la experiencia de un vuelo real. Lo mismo pasaba con las tiendas de Apple. Se creaban varias "tiendas" y Steve Jobs iba a ellas a "sentir" la experiencia. Basado en esto cambiaban ciertos aspectos, mezclaban diferentes ideas y despreciaban otras, hasta tener la tienda a la que se quería llegar.

Estrategias para obtener retroalimentación

Existen muchas estrategias para obtener comentarios y retroalimentación de los usuarios.

Por ejemplo, **enviar encuestas**. Es una muy buena forma de obtener *mucha* retroalimentación pero los resultados que se obtienen no son muy precisos. Los humanos tendemos a "mejorar" cuándo estamos llenando una encuesta. Muchas

veces al preguntarnos por un servicio somos más que compresivos y calificamos mejor de lo que recibimos. Lo mismo cuándo nos auto-calificamos, nadie quiere sentirse muy mal por su desempeño, entonces lo normal es subir un poco la calificación que nos damos. Estos comportamientos hacen que las encuestas sean una excelente manera de tener *mucha retroalimentación* pero no *muy buena retroalimentación* (y por buena me refiero a real, no a positiva).

Otra forma de obtener retroalimentación es a través de la **observación de los usuarios**. Viendo como los usuarios trabajan con el prototipo nos puede ayudar a mejorar puntos claves en el diseño y la experiencia al rededor del producto que deseamos crear. Por ejemplo, si hacemos un producto para que los camioneros utilicen mientras manejan y tiene teclas o botones pequeños, es muy probable que esto dificulte su uso, pues muchos camioneros utilizan guantes grandes para el frío.

Esta técnica de observar a las personas, se basa en la forma como los antropólogos estudian culturas. Donde primero observan y posteriormente forman parte de la cultura. La idea de esta técnica es poder obtener el conocimiento tácito que tienen las personas. Se busca saber:

- ¿Qué hacen las personas actualmente?
- ¿Qué metas u objetivos tienen las personas (con el producto)?
- ¿Cómo estos objetivos o actividades que se quieren realizar encajan en un ecosistema más grande?
- ¿Qué similitudes o diferencias hay entre las personas?
- El contexto en general, como ¿a qué horas usarían el producto? ¿Con quién? ...

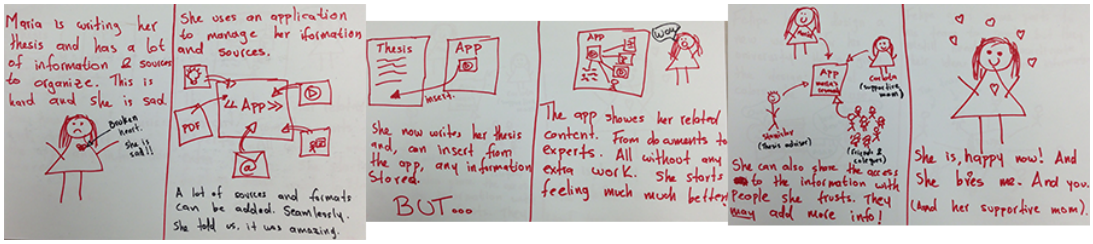
Cuándo estamos desarrollando un producto, es importante tener claro que se quiere realizar algo que las personas quieran o necesiten, pero esto **no** es (necesariamente) desarrollar lo que las personas piden. Muchas veces las personas *no saben qué quieren*. Esto es aún más cierto en tecnología. Uno **no debe estar pendiente de lo que la gente dice, sino de lo que hacen**.

Prototipos Rápidos

Como se ha mencionado previamente, a medida que el desarrollo del producto avanza, los prototipos van siendo más y más "fieles" a lo que será el producto final, y por eso en cada etapa del desarrollo se deben utilizar herramientas que nos permitan mostrar el estado donde estamos. En un principio no tiene mucho sentido tener un *Wireframe* en HTML. Miremos el proceso que se puede seguir para realizar una aplicación web.

Storyboards

Un *Storyboard* es un conjunto de dibujos pequeños, que se utilizan para mostrar el **escenario** donde se usará la aplicación. Por ejemplo:



Es muy importante resaltar que el *Storyboard* quiere contar una historia pero no quiere decir *cómo será* la aplicación. En ningún momento es necesario (y no es recomendable) empezar a pensar en la interfaz de usuario en un *Storyboard*. El **Storyboard es para mostrar las tareas y escenarios que se quieren soportar con la aplicación**.

Uno de los problemas más grandes y comunes al hacer diseño de interfaces de usuario, es concentrarse en la interfaz **antes** de concentrarse en la tarea que se quiere resolver. El *Storyboard* busca resolver esto al incluir personas usando la aplicación.

Un *Storyboard* debe tener:

- **Escenario:** ¿Quiénes están involucrados? ¿En qué ambiente? ¿Qué tarea quieren desarrollar?
- **Secuencia:** ¿Qué pasos deben dar? ¿Qué situación haría que las personas utilizaran la aplicación? ¿Qué tarea se está mostrando?
- **Satisfacción:** ¿Qué motivó a las personas a usar el sistema? ¿Qué le

permite a las personas realizar el sistema? Y más importante, ¿qué necesidad resuelve el sistema?

Nuevamente, lo más importante de un *Storyboard* es que permite pensar/mostrar cómo resolveríamos un problema de los usuarios, sin pensar aún en la interfaz que tendríamos.

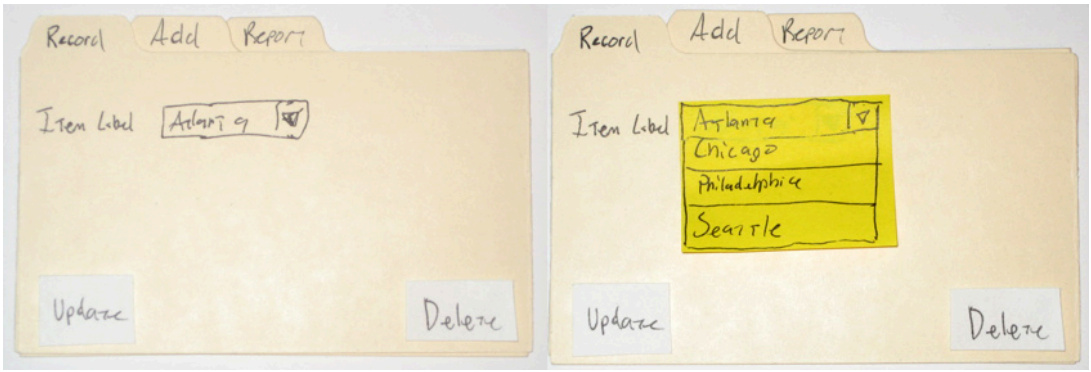
Aquí se puede encontrar una guía para hacer *storyboards*.

Prototipos en Papel

Una vez los escenarios estén claros, se puede pasar a la primera etapa de la construcción de las interfaces. Hacer prototipos en papel. La idea en esta etapa, así como en los *Storyboards* es poder tener resultados poco fieles pero que se puedan construir rápidamente. La diferencia en esta etapa es que **ya** se empieza a pensar en la interfaz del producto.

Estos prototipos no necesariamente tienen que ser completamente hechos a mano. Uno puede imprimir imágenes de elementos comunes del sistema operativo (un botón de iPhone por ejemplo) que ayudará a dar mucho más contexto al usuario de dónde se utilizará la aplicación.

A List Apart, tiene un muy buen artículo sobre prototipos en papel para páginas web.



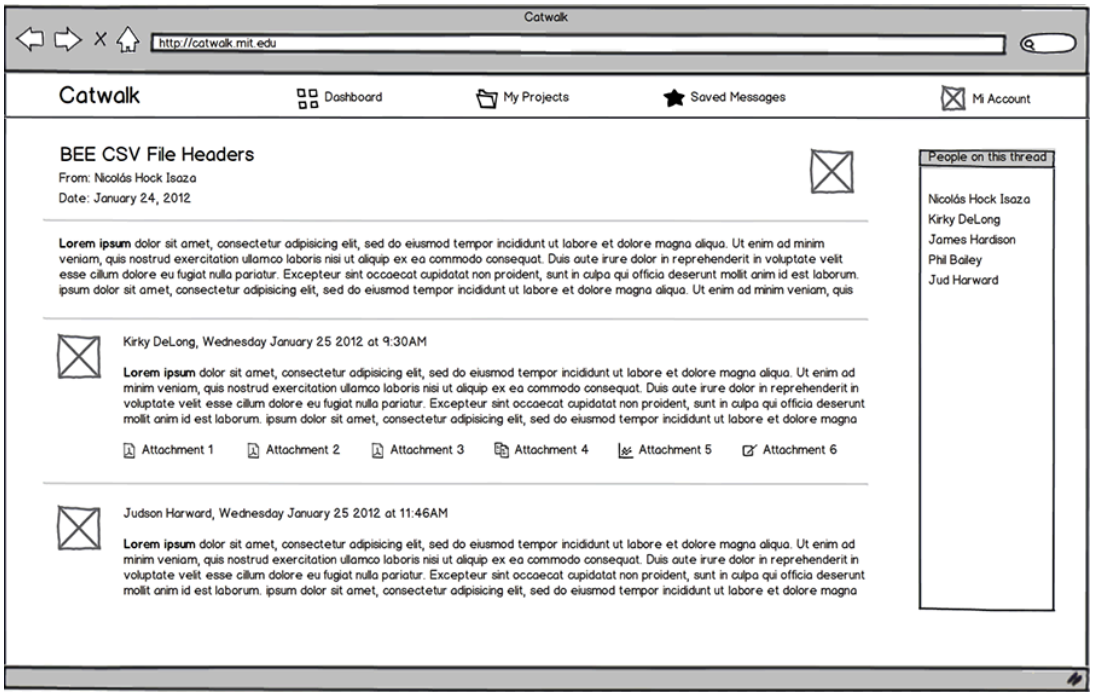
En esta etapa se puede cambiar toda la arquitectura y diagramación de la solución que se desea construir. Se pueden explorar más escenarios para

después pasar a archivos digitales.

Maquetas Digitales

Una vez se tenga buena retroalimentación con los prototipos en papel, se puede pasar a prototipos un poco más fieles utilizando maquetas digitales (*mockups*). Estos prototipos toman mucho más tiempo que los de papel, por eso es importante hacerlos después, pues ya se ha tenido retroalimentación que puede prevenir errores que tomarán más tiempo (tanto en hacerlos como en corregirlos).

Hay muchos programas para hacer estos prototipos. Uno muy conocido y muy bueno es Balsamiq. Aquí hay una muestra de una aplicación para almacenar discusiones que se han tenido mediante correo electrónico.



Aquí, también se pueden crear varias diagramaciones de la solución que se quiere implementar. Como se puede ver en la imagen, la fidelidad aún es baja, pero ya se empieza a pensar en la **estructura** de la interfaz.

A partir de este punto se puede utilizar una estrategia llamada **Mago de Oz**, donde se muestran los prototipos a usuarios "reales" pero las funciones más

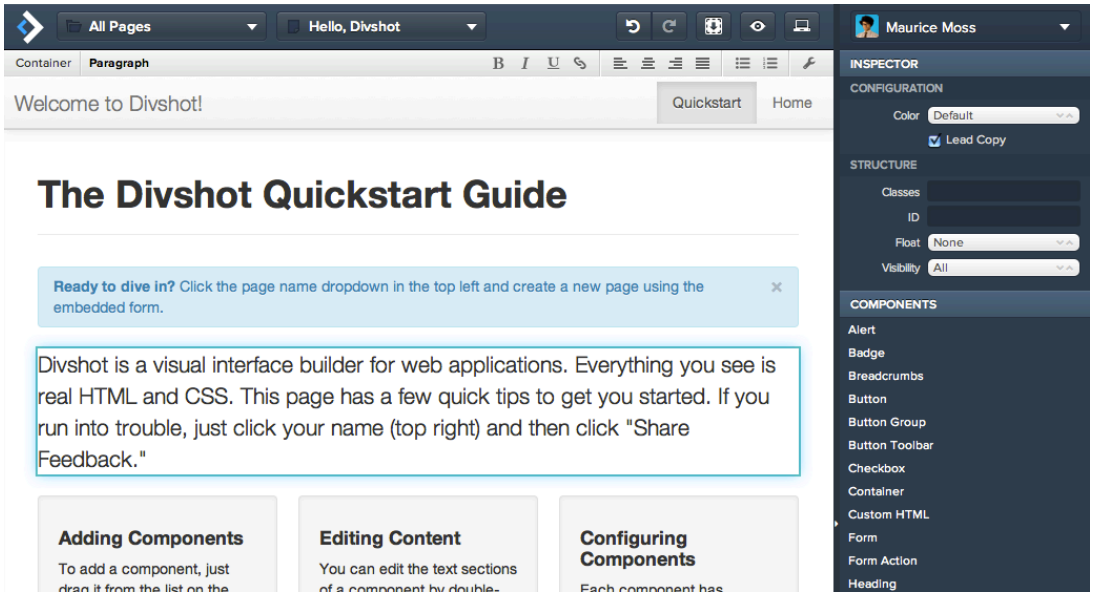
complejas del sistema se simulan. Por ejemplo, encontrar la ruta más corta en un mapa, teniendo en cuenta el tráfico y el reporte del tiempo. Uno puede decirle al usuario "esta es la ruta", sin calcularla. Pues lo importante no es como se calcula sino lo que el usuario hace en ella.

Como en el Mago de Oz, todo era mentira y había alguien "detrás de la cortina". Se debe buscar que tener alguien detrás de la cortina para que la aplicación **parezca y se sienta real** aunque no lo sea.

HTML Estático

Tener un prototipo en HTML estático es una tarea que se puede hacer cada día más fácil con la ayuda de herramientas, pero, aún así, es costoso comparado con las etapas previas de diseño. En esta etapa ya se empiezan a variar aspectos "menores" del diseño, como colores y tipografía.

Para crear un HTML estático se recomiendan herramientas como DivShot. Con esta herramienta se construyen las páginas HTML en una interfaz muy fácil de utilizar y sin la necesidad de codificarla.



Dynamic Data & Databases

El último paso del proceso sería la persistencia y muestra de datos reales. Teniendo ya la interfaz lista, podemos pensar en cómo se van a guardar los datos y cuáles serán "las reglas de la aplicación". En este punto se deben tener más que claras las interacciones de los usuarios con la aplicación lo que facilita mucho codificar y modelar el problema desde un punto de vista más técnico.

Creando y Comparando Alternativas

Esto se basa en un estudio de Steven Dow en Stanford, donde se planteaba si, en el momento de diseñar, es mejor centrarse en la **cantidad o en la calidad**? Se ilustra el escenario de una profesora que dividió la clase en dos partes, a la primera mitad se le iba a calidad exclusivamente sobre la cantidad de cosas que hicieran, la otra mitad iba a tener su calificación sobre la calidad de una pieza.

El resultado fue que, mientras la mitad "de cantidad" estaba ocupada haciendo muchas cosas -y aprendiendo de sus errores-, la mitad de la calidad se pasó mucho tiempo pensando y buscando teoría para soportar su calidad. Al final salieron con mucha teoría, pero nada de trabajo.

Esto nos indica la importancia de tener prototipos fáciles de cambiar (o desechar por completo), y varias alternativas para esto. Esto se hace para prevenir lo que Karl Duncker llama "Fijación Funcional", esto consiste en atacar una y solo una idea (normalmente la primera que tenemos) hasta que la hagamos funcionar (si podemos).

El tener una sola idea, y alguien la critica, es mucho más probable tomar esa crítica como "personal" a si tengo varias ideas y critican alguna. Además, al crear diferentes alternativas, se puede transferir el conocimiento aprendido en una hacia las otras y así evolucionar el diseño aún si se trata de una idea diferente.

Es importante resaltar que no es solo crear varias (y mostrar la mejor), es crear varias y mostrar varias. Conseguir retroalimentación de todas para evolucionar de acuerdo a todas las opciones, no a la que consideramos mejor.

Evaluación con Heurísticas

Esta evaluación fue inventada por Jakob Nielsen y busca encontrar problemas de usabilidad en el diseño. Esta técnica busca obtener retroalimentación muy rápido y se puede utilizar tanto en productos terminados como prototipos (papel, digitales o HTML estáticos-más fieles).

La idea es que se entrega a las personas que van a evaluar el diseño, una serie de principios que deben ser utilizados para evaluar el diseño. Cada persona lo hará de manera independiente y después compartirán los resultados. Nielsen propone diez heurísticas:

1. Visibilidad del estado del sistema: El sistema debe *siempre* informar al usuario de lo que está pasando. Esto se puede lograr a través de retroalimentación (mensajes mostrando que una operación fue exitosa, o falló) en el momento adecuado.

2. Consistencia entre el mundo real y el sistema: El sistema debe hablar el lenguaje del usuario. Con palabras, términos y conceptos similares a los del usuario (en el área de la solución) y no técnicos enfocados hacia el sistema. Se deben seguir las convenciones del mundo real, mostrando la información en un orden lógico y natural.

3. Control y libertad del usuario: El usuario debe poderse "devolver" en el sistema cuando, por ejemplo, presiona un botón indeseado. Esto se debe permitir sin la necesidad de hacer muchas cosas.

4. Consistencia y estándares: Las cosas que signifiquen lo mismo, se deben llamar por el mismo nombre. Si dos acciones son la misma, el usuario debe saberlo solo con el nombre.

5. Prevención de errores: Aún mejor que buenos mensajes de errores es tener un diseño que prevenga los errores. Si se tiene un formulario con campos requeridos, mostrarlos al usuario y alertar (sin dejarlo seguir) sobre los campos requeridos que no ha llenado.

6. Reconocimiento y no recordar: El sistema debe minimizar la necesidad del usuario de recordar cosas. Ofrecer ayuda constante mientras se utiliza la

herramienta, tener instrucciones siempre visibles o un orden lógico de la información son algunas de las formas para lograr esto.

7. Flexibilidad y eficiencia de uso: Tener comandos (invisibles para los usuarios principiantes) que permitan realizar las tareas de una manera más fácil y rápida, pueden mejorar mucho la experiencia del usuario en la aplicación. Sobre todo para acciones que se repiten mucho.

8. Estética y diseño minimalista: Los mensajes o menús, no deben contener texto que sean irrelevantes. Cada pedazo de información innecesaria compite con la información realmente necesaria por la atención del usuario.

9. Ayudar a los usuarios a reconocer, diagnosticar y corregir los errores: Los mensajes de error deben ser en lenguaje escrito (usando la heurística 2) y no con códigos del sistema. Deben presentar una opción para solucionarlos.

10. Ayuda y documentación: Además de las ayudas "en línea" (heurística 6) se debe tener un buen sistema de ayuda del sistema completo. Un lugar donde los usuarios puedan buscar cómo usar el sistema y preguntar por acciones que no logren realizar. Este sistema de ayudas debe ser **muy fácil de utilizar**, donde el contenido se pueda ser **buscado con facilidad**.

Estas diez heurísticas son sacadas de la página del Nielsen Norman Group donde adicionalmente vinculan a una página con ejemplos de cada una de las diez heurísticas en páginas web.

Como dice Nielsen, estas son heurísticas por que son reglas naturales y no unas guías específicas de usabilidad.

Personalmente me asombra que las diez reglas son muy intuitivas. Tanto que al leerlas decimos "obvio, ¿qué sistema no tiene esto?", pero al momento de construir soluciones las olvidamos casi por completo!

Adicionalmente a las diez heurísticas de Nielsen, se pueden agregar varias que sean específicas al contexto de la solución.

Para hacer una evaluación heurística, se deben tener varios evaluadores. Se les

debe explicar el dominio de la solución (conceptos y términos médicos por ejemplo) y un poco de contexto de como se va a utilizar (o como se espera que se utilice) la solución.

Después cada evaluador realiza la prueba. **Por separado**. Y al final se comparten resultados. En esta evaluación se debe poner una calificación a cada error encontrado (crítico, severo…) que después se puede complementar con la calificación del grupo.

Al final, se analizan los resultados entre todos, evaluadores y desarrolladores.

Manipulación Directa del Producto

Cuando se realiza un producto con el cual las personas interactúan, la observación es clave para obtener retroalimentación sobre como las personas utilizan una solución similar o un prototipo. Para entender más la forma de utilizar, se tienen seis preguntas: **¿Qué tan fácil puede alguien

- determinar o definir la función del dispositivo?
- encontrar que acciones se pueden realizar con el dispositivo?
- determinar los pasos necesarios para "lograr el objetivo" con las acciones necesarias que necesitan realizar?
- realizar la acción (o acciones)?
- saber en qué estado está el sistema?
- basado en el estado que está el sistema, saber en qué paso estoy?

*Preguntas propuestas por Don Norman en **The Design of Everyday Things***

Para reducir abismos entre la intención que tienen los desarrolladores del producto y la forma como éste es utilizado por los usuarios, se debe proveer:

- **Visibilidad:** Si existe algo que el sistema puede hacer, mostrárselo al usuario. Si es un producto físico, se puede utilizar un botón o empuñadura.
- **Retroalimentación:** Mantener al usuario siempre informado sobre lo que está pasando y lo que puede hacer.
- **Consistencia:** Nombrar consistentemente los elementos del sistema. Dar

nombres comunes para el usuario.
- **Operaciones no destructivas:** Permitir al usuario "des-hacer" una acción sin la necesidad de reiniciar el proceso completo.
- **Discoverabilidad:** Todas las funciones del producto se deben poder descubrir al navegar los menús del producto. En un producto físico al abrir tapas (como en un control remoto).
- **Fiabilidad:** El producto debe funcionar. Sin excusas. Y las operaciones u eventos no se deben dar de manera aleatoria.

Se podría resumir que los aspectos importantes de un producto cuando el usuario interactúa directamente con éste son:

- Retroalimentación Instantánea
- Representación estandarizada de objetos a lo largo del producto
- Utilizar metáforas de elementos conocidos por el usuario para referirse a objetos en el sistema (como el Escritorio en un sistema operativo)

Experimentos en la Web

Como en todos los productos, en las páginas web se pueden probar diferentes alternativas para elegir la mejor. Esto es conocido como *A/B Testing*. Se tienen dos (o más) alternativas y se divide el tráfico entre ellas, se toman métricas y se elige la mejor.

Muchas veces los cambios en diseño tienen un impacto grandísimo en las acciones de los usuarios en una página Web. Solo cambiar el texto de un título o un botón puede influir en que más (o menos) usuarios se subscriban a la página. A/B Testing es una forma rápida de obtener retroalimentación de usuarios reales.

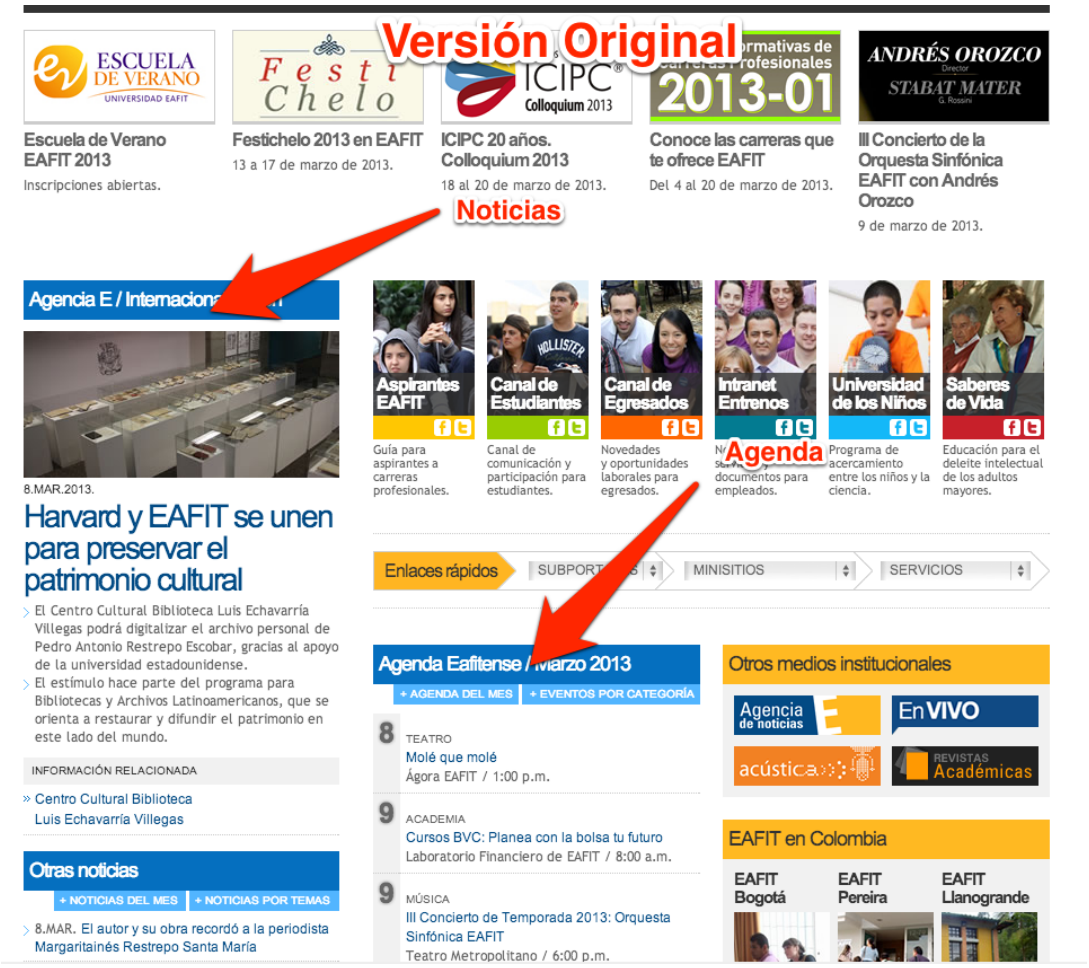
Los experimentos deben tener un objetivo, por ejemplo, que el usuario se subscriba a los correos masivos, o compre un producto. Estos objetivos se llaman "acciones". Los botones o elementos que invitan a estas acciones se llaman "call to action" (llamado a la acción). Es muy común variar atributos como color o texto dentro de estos "call to action".

Se da un ejemplo de la campaña de Barack Obama, donde cambiar el texto del botón principal de "Sign Up" por "Learn More" aumentó el número de personas llenando el formulario en un 18%.

Existen muchas opciones para desarrollar pruebas A/B. Una muy recomendada es Optimizely que permite editar una página específica sin necesidad de tener conocimientos Web. Google Analytics también permite llevar a cabo experimentos a través del *Google Website Optimizer*.

Un ejemplo con la página de EAFIT.

Versión Original



Versión Editada

Una de las grandes ventajas de las pruebas A/B es que la subjetividad del equipo de trabajo no tiene lugar en el resultado final. Por más que los diseñadores piensen que una opción es la mejor, no se puede alegar contra los datos. Al final, lo importante no es lo que el equipo piense que es mejor, sino lo que funciona mejor con los usuarios.

Comentarios del Curso

De todos los cursos que he hecho en Coursera, creo que este es al que más tiempo le han dedicado. Los vídeos son bien editados y durante los vídeos el profesor "escribe" sobre la presentación para explicar los conceptos. Pero, también es el curso que más trabajo y dedicación requiere por parte de los

estudiantes. Esto no es necesariamente malo, más aún si es un curso "por créditos".

Los vídeos y ejemplos eran de proyectos reales, con prototipos de productos conocidos y en contextos muy conocidos, lo cual invitaba a investigar más sobre el producto que mostraban.

Evaluación Por Pares

Adicional a los exámenes y quizzes, se tenía un proyecto que se realizaba a lo largo del curso. Cada semana se debía entregar una parte más del proyecto. Aquí es donde más trabajo se requería.

El proyecto era complejo y había muchas actividades para hacer. La actividad a realizar como tal no era muy ligada a los vídeos que se veían en la semana, lo cual complementaba el aprendizaje (pero requería aún más dedicación) pero lo más difícil era que había que evaluar otros proyectos también.

Antes de evaluar a un compañero, uno debía pasar un proceso de "entrenamiento" para evaluar la actividad de la semana. Este proceso, consistía en evaluar entregas de cursos anteriores. Lo malo (o bueno - tengo sentimientos encontrados) de este proceso es que si uno no calificaba "igual" al profesor, uno estaba calificando mal y el proceso de entrenamiento era eterno.

Así, el tiempo que uno tenía reservado para evaluar, se iba en "entrenamiento". Y si uno no evaluaba, le bajaban la nota en un 20%.

Creo que un poco más de retroalimentación en la parte de "entrenamiento" era necesaria. Me gusta la idea de calificar otros trabajos, pero la fase de entrenamiento era irritante.