Porqué los proyectos informáticos demasiadas veces se convierten en una pesadilla

Anatomía de un Fracaso

Por Daniel Mordecki

Los proyectos de tecnología ya son famosos por sus desviaciones en presupuesto y plazo al punto de que muchos gerentes del negocio evitan embarcarse en ellos. Probablemente, estemos necesitando un enfoque completamente nuevo.
9 de abril de 2002

Tal vez este artículo no sea para usted.

Los proyectos informáticos que exceden en un 200 o 300 por ciento su plazo original son tan frecuentes, que se han transformado en algo mítico dentro de la industria informática. Las áreas comerciales han asumido que se trata de una realidad inmodificable y los gerentes de estas áreas sufren impotentes cuando no tiene más remedio que embarcarse en un nuevo proyecto que hace uso intensivo de la tecnología de información, algo que sucede muy a menudo en el mundo de hoy.

Mientras las revistas de tecnología y de negocios cantan loas a proyectos fantásticos, donde se ganan decenas de millones de dólares, la realidad que cada uno de nosotros vive día a día es muy distinta. La tecnología se ha vuelto imprescindible y omnipresente, pero el control y gerenciamiento de los proyectos está lejos de ser satisfactorio. Se trata de una fuerza irresistible, que todo lo llena, que todo lo abarca, pero que a menudo queda totalmente fuera de control, consumiendo todos los recursos y arrasando todo a su paso.

Si le parece que la descripción es exagerada, entonces esta nota no es para usted, sin duda alguna. Este artículo está dirigido a encontrar a aquellos que por haber vivido las situaciones que se describen se identificará de inmediato. A todos aquellos que se sienten tanta

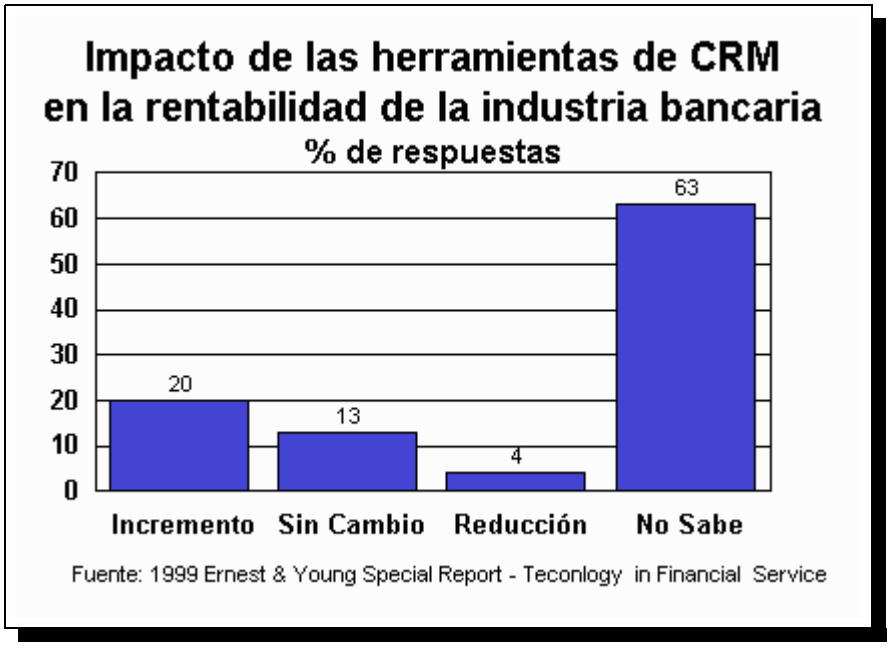

Figura 1 - No es común encontrar en revistas de negocios estadísticas como ésta. Mucho más común es encontrar historias de éxito fantástico, contadas por los propios interesados en mostrar ese éxito.

impotencia frente a estas situaciones como cuando pierden un archivo, cuando se cuelga su equipo o cuando la impresora escupe sin cesar papel inutilizado.

Este artículo está dirigido todos aquellos que se sienten tanta impotencia con los proyectos informáticos, como cuando pierden un archivo, cuando se cuelga su equipo o cuando la impresora escupe sin cesar papel inutilizado.

Parte I - Las bases del problema

Comparada con disciplinas milenarias, como la arquitectura, la medicina o la física, la informática es una disciplina aún en pañales. Pero ninguna disciplina consiguió en 50 años invadir prácticamente todos los aspectos de la vida sobre la tierra. Además de las computadoras, hoy todo tiene microprocesadores y software dentro: el televisor, el auto, la heladera, el reloj y el termómetro. Y todo comienza a comportarse de una forma incómoda, insensata, que nos hace sentir torpes y tontos.

Se trata de una disciplina absolutamente inmadura, que en su crecimiento arrollador ha tomado prestado de aquí y de allá metodologías, técnicas y soluciones, sin darse a tiempo a generar las propias en numerosas áreas, abriendo así flancos enteros a los problemas y el descontrol.

El problema se agrava porque el software es tal vez uno de los elementos más complicados con los que le ha tocado interactuar a la mente humana. Se trata de un intangible basado en un modelo lógico matemático estricto, cuya expresión exterior nada tiene que ver con su estructura interna y que no responde a las leyes físicas a las que los humanos estamos acostumbrados.

el software es tal vez uno de los elementos más complicados con los que le ha tocado interactuar a la mente humana

Esto crea problemas nunca antes afrontados, como por ejemplo la revisión del trabajo de sus subordinados. En cualquier área, un gerente es capaz de supervisar a sus subordinados, verificar que hacen sus tareas correctamente. Sin embargo, lleva más tiempo entender el código generado por un tercero que el que llevó escribirlo o directamente codificar el programa nuevamente. Eso hace que en la práctica, nadie revise el código que se escribe para un proyecto. Se diga lo que se diga, se negocie lo que se negocie, sus programadores conservarán siempre la libertad de escribir el código de la forma que deseen, sin que nadie vaya jamás a controlarlos.

De todo este panorama se derivan situaciones extremadamente complejas, en las cuales gerentes no cuentan con herramientas mínimas para manejarse. Gerentes que habitualmente toman decisiones sobre todas las áreas de su empresa, se sienten en la desolación total ante las decisiones sobre tecnología informática. La base para la generación de esas herramientas ya está puesta, y está ahora en cada empresario buscarlas, entenderlas y aplicarlas en su empresa.

La lista de features

Usted puede detectar peligro de fracaso o un proyecto que se saldrá de su cauce, cuando el proyecto nace con una lista de features o características: la lista de lo que el aplicativo debe hacer.

La lista de features nace a partir del corazón del modelo de desarrollo de software. Un feature no es otra cosa que un trozo de código que desarrolla una tarea determinada, y un aplicativo no es otra cosa que la unión de todo ese código para ser ejecutado junto. Desde los primeros días los programadores tienen la necesidad en cada proyecto de descomponer su trabajo en partes de menor tamaño, codificarlas y luego unirlas. Es lo que se denomina Análisis de Sistemas. Y a falta de mejor oferta, han ofrecido este método para describir como lucirá el producto final.

Pero dado que en el software, la forma externa es absolutamente distinta que el modelo interno, la estrategia de ir desde la superficie hacia el corazón del aplicativo no alcanza para

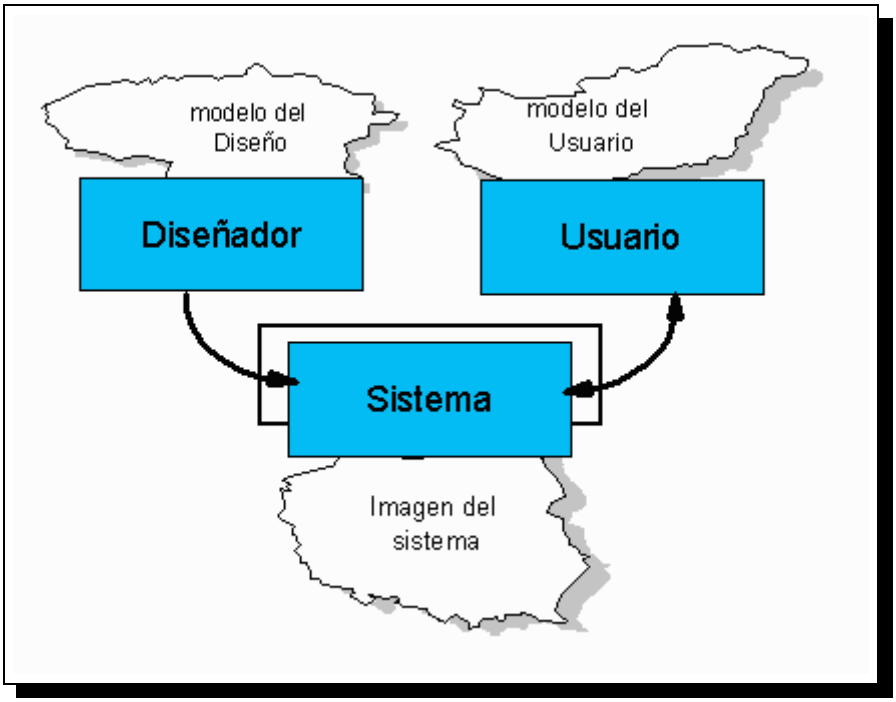

Figura 2 - Modelos Conceptuales. El *modelo del diseño* es el modelo conceptual del diseñador. El *modelo del usuario* es el modelo mental desarrollado a través de la interacción con el sistema. La *imagen del sistema* resulta de la estructura física que se ha construido (incluyendo documentación, instrucciones y etiquetas) El diseñador espera que el modelo del usuario sea idéntico que el modelo del diseño. Pero el diseñador no habla directamente con el usuario, sino que la comunicación se realiza a través de la imagen del sistema. Si la imagen del sistema no deja en claro el modelo del sistema de forma consistente, el usuario generará un modelo mental equivocado.

Donald A Norman - The Design of Everyday Things - 1988

especificarlo. No es que el análisis no sea necesario, es imprescindible. Pero no para definir el producto final sino para desarrollar el trabajo estrictamente informático de desarrollo.

Modelos Conceptuales

Para que una especificación sea capaz de sentar las bases de un proyecto, no solo debe abarcar el modelo interno del software, lo que Donald A. Norman denomina el Modelo del Diseño, sino además el Modelo del Usuario, es decir, la representación conceptual que el usuario generará en su mente de como funciona el sistema, y de como alcanzará sus objetivos al usarlo. Para ello es imprescindible cubrir la Imagen del Sistema al especificarlo, dado que ésta es quien opera como nexo entre lo que está en la mente del programador hoy y lo que estará en la mente del usuario cuando el producto esté terminado.

Los seres humanos tenemos la tendencia y necesidad natural de generar modelos de como funcionan las cosas que nos rodean, los objetos y sistemas con los que interactuamos. No importa el nivel de exactitud de ese modelo: siempre existirá uno, es imprescindible para poder actuar. Si no entendemos como se genera el viento, creeremos entonces que cuando muchas hojas de un árbol se mueven a la vez generan un poquito de viento que a su vez moverá otras hojas y así sucesivamente hasta generar una tempestad si la suerte nos lo depara. No importa lo irracional, no importa la violación de las leyes de la física: funciona. Se mueven las hojas, hay viento. Se mueven más, más viento. Se mueven mucho, mucho viento. Pero cuanto más adecuado es el modelo mental del usuario, más probable es que consiga sus objetivos al interactuar con un objeto o sistema. A nadie en su sano juicio se le ocurriría sacudir los árboles para generar energía eólica.

Una lista de características, independientemente del nivel de detalle con el que se la elabore, no alcanzará jamás para describir un producto basado en software.

La especificación deber ir entonces desde la superficie del sistema hacia el Modelo del Usuario, describiendo los objetivos de éste y como los alcanzará utilizando el producto, la Imagen del Sistema opera como vínculo en esta tarea.

Una lista de características, independientemente del nivel de detalle con el que se la elabore, no alcanzará jamás para describir un producto basado en software.

La determinación de los plazos

Una vez que se generó la lista de features, se procede a la determinación de los plazos. Para ello, al lado de cada feature se anotan las horas o días que llevará programarlos. En este momento, los programadores sentenciarán los features que más les desagradan, y premiarán los que más simpáticos les resultan o los que ellos propusieron. Alcanza con poner más horas o días de trabajo, para aumentar la probabilidad de que un feature desaparezca de la versión final. Por el contrario, aquellos features que el programador considera útiles y que se pueden incorporar “sin esfuerzo” llevarán un cero al lado y serán incluidos sin discusión.

A la suma de todo los tiempos necesarios, se le agrega un plazo razonable para poner todas las partes del código a funcionar juntas y testear el sistema, y el resultado final coincide exactamente con la fecha en la que hay que entregar el sistema.

La determinación de plazos basada en la lista de features es mala por su efecto secundario, el de ser la única ponderación de prioridades del proyecto.

No importa si esta tarea se lleva adelante con un lápiz y un papel o con un sofisticado programa de manejo de proyectos, el resultado es el mismo: se califica la importancia de cada feature, con un ponderador muy fuerte como el tiempo y dinero que costará producirlo, independientemente de la importancia que tiene para que el sistema cumpla con sus objetivos. Dado que no existe otra lista de prioridades, esta lista será la única base y determinará sin duda alguna el resultado final. Por ejemplo, si en un proyecto de seis meses, el sistema de ayuda contextual tiene una asignación de 300 horas/hombre de programación y la calculadora en línea 20 horas, el sistema tendrá sí o sí calculadora y probablemente en algún momento de la negociación de features perderá la ayuda contextual.

La determinación de plazos basada en la lista de features no es mala por su resultado primario, establecer el cronograma del proyecto, sino por su efecto secundario, el de ser la única ponderación de prioridades del proyecto.

La negociación de features

Apenas comienza el proyecto, comienzan a salir a luz la diferencia entre los modelos conceptuales que generaron los distintos participantes de su definición. La idea del Gerente de Marketing era distinta de la del Gerente Comercial y éstas dos distintas de la del Gerente de Sistemas. Inclusive en un proyecto muy grande, dentro del área de sistemas pueden surgir ideas completamente contrapuestas de como será el sistema al final.

Y lo que al principio son malentendidos menores, aparentemente sin importancia, poco a poco se transforman en discrepancias mayores, que generan roces, enfrentamientos y recriminaciones. Para que el sistema haga “lo que debería hacer” será necesario escribir de nuevo código ya escrito, algo que está último en las preferencias de cualquier programador.

Los argumentos más comunes son la obviedad de la necesidad de determinada funcionalidad o característica y su contra cara, que determinada área o departamento no expresó a tiempo lo que quería. Dicho de otra forma, un departamento le echa en cara a Sistemas que no hizo algo obvio, y éste le contesta que no lo hizo porque no fue incluido oportunamente en la lista de features. Se negociará entonces una nueva lista de features, mutación de la anterior, que contenga el feature deseado, con un nuevo numerito de horas necesarias a su lado. Si bien al principio Sistemas absorbía los cambios en el cronograma general, rápidamente la negociación incluirá decidir entre una de dos opciones: eliminar otro feature o correr la fecha final.

Dado que en su momento se calificaron los features con el costo en horas/hombre, los más pesados en trabajo serán los primeros candidatos a desaparecer, saliendo a luz la importancia de la determinación de plazos.

El proceso discusión sobre la modificación de la lista de features durará hasta el final del proyecto sin interrupciones, de forma continua. A medida que avanza el proyecto cada vez hay más código, cada vez son más costosos los cambios, pero cada vez se hacen más evidentes las lagunas que la lista de features dejó a la hora de definir el producto final y generar un consenso sobre como sería este producto. Junto con ello, subirá la tensión en las discusiones, las acusaciones, los bandos y la generación de un ambiente negativo.

La negociación de features es tal vez el fenómeno más nefasto en el desarrollo de un proyecto informático.

si no se desea cancelar totalmente el proyecto, se decidirá que en determinada fecha, pase lo que pase, se dará por terminado.

La finalización del proyecto

Si usted no sabe hacia dónde va, jamás podrá saber si llegó. Dado que no hay una clara definición del destino, será imposible determinar cuando termina el proyecto, por lo que no hay manera de alcanzar el fin. Parece paradójico, o ridículo, pero es real.

En algún momento del transcurso del proyecto, comienza a hacerse evidente que la fecha de finalización no solo se corrió demasiado en el tiempo, abarcando dos o tres veces el tiempo y el presupuesto originales, sino que es tan inalcanzable como el horizonte. La permanente y desgastante negociación de features, aleja cada vez más el borroso final del proyecto. En algún momento, si no se desea cancelar totalmente el proyecto, se decidirá que en determinada fecha, pase lo que pase, se dará por terminado..

Esto tiene una primera consecuencia: la determinación de los plazos hará su trabajo final, dado que a medida que se llega a la fecha límite, caen por su propio peso los features más costos que lograron sobrevivir la negociación de features.

La segunda consecuencia es el recorte automático del sistema de ayuda, de la documentación del usuario y de la documentación del sistema. Probablemente se sume a éstas alguna otra variable de ajuste.

Lo que comenzó con toda pompa, termina de forma abrupta, sin dejar satisfecho a nadie, sin cumplir los objetivos principales y dejando tras de sí una secuela de heridas que sin duda saldrán a la luz en el próximo proyecto.

La negociación de features es tal vez el fenómeno más nefasto en el desarrollo de un proyecto informático.

Parte II - Un enfoque distinto

Afortunadamente existe un enfoque completamente distinto para la definición y el gerenciamiento del proyecto, que se basa en incluir una etapa de diseño antes de la codificación. El diseño es una etapa completamente nueva en los proyectos informáticos, que se desarrolla antes de comenzar a codificar el software. En el artículo “Los objetivos del diseño” se expone una definición completa y un detalle de los objetivos a alcanzar con el diseño.

asuma que el código es como el hormigón, una vez que está hecho es extremadamente costoso dar marcha atrás

Este enfoque apunta a eliminar las causas que conducen en los proyectos a distorsiones y fracasos, proponiendo técnicas y conceptos nuevos.

Diseñar primero, codificar después

Supongamos que en el desarrollo de un sistema el cronograma de 10 meses está dividido en 6 meses de diseño y 4 meses de codificación. No se necesita buscar un consultor demasiado experiente, todos dirán al unísono que es una locura. ¿Para qué perder 6 meses de tiempo? se puede comenzar a trabajar ya y diseñar la interface al final. ¿Porqué no empezar ya a desarrollar aquellas partes del sistema que no dependen del diseño? ¿Que es lo que hay que diseñar durante tanto tiempo? Lo que se diseñe durante 6 meses, seguro que no se va a poder codificar en 4.

Ahora supongamos que se trata de la construcción de un edificio, y que el arquitecto determina que llevará 6 meses el desarrollo del proyecto y 4 meses la obra. Nada más sensato. El caso contrario indica que dado que el edificio tendrá 10 pisos, ya se puede ir haciendo el pozo, porque por lo menos habrá que excavar 3 o 4 metros, y comprando los materiales, porque hormigón, hierro, arena, etc. se va a consumir. Un despropósito.

Para simplificar las ideas, asuma que el código es como el hormigón, una vez que está hecho es extremadamente costoso dar marcha atrás. No hay ninguna tarea de codificación que no dependa del diseño. Una vez que se escriba la primera línea de código, la estructura del sistema estará determinada, y si esta no coincide exactamente con el objetivo final del proyecto, la semilla del fracaso habrá dado su primer brote.

En el artículo “Diseño: usuarios al poder” se expone en detalle una metodología de diseño específica para proyectos informáticos. Lo más destacable a los efectos del presente artículo es señalar que el diseño determina el modelo conceptual del aplicativo, describe a cabalidad el producto final y genera los consensos necesarios para poder llevar adelante el proyecto con éxito.

La determinación del modelo conceptual en contraposición a la lista de features, permite a las partes que trabajan hoy en la creación de un aplicativo y en el futuro a los usuarios del mismo, entender como alcanzar sus objetivos al utilizarlo. No se trata de lo que el software hace, o de lo que el software tiene, sino de como interactúa un ser humano con la aplicación para completar las tareas necesarias a la hora de cumplir un objetivo concreto. De esta manera quedará especificado el comportamiento del sistema y las prioridades de las distintas partes dentro de él.

Consecuencia inmediata es la capacidad del equipo de trabajo de definir el producto final, de cómo lucirá el producto terminado. Ya dijimos que el software es un intangible, de alto nivel de abstracción, cuya expresión externa es completamente distinta del modelo lógico interno. Es necesario utilizar todas las herramientas disponibles para describir como se comportará el software. La lista de features puede tal vez ser una de ellas, pero no dude en utilizar prototipos, story boards, diagramas de flujo, descripciones escritas, dibujos de las pantallas, animaciones y en general todo lo que existe o pueda inventar, para llegar a que todos entiendan a cabalidad

como lucirá el aplicativo una vez terminado y como interactuarán los usuarios con el sistema cuando éste esté en producción.

Solo en este momento, el equipo estará preparado para escribir la primera línea de código. Ahora está completamente determinado el proyecto, y la codificación será una tarea lineal en pos de un objetivo común y compartido. Si el diseño está bien hecho, no habrán sorpresas con los resultados, y se evitará completamente la necesidad de negociar permanentes cambios al alcance del proyecto.

Jamás viole esta ley: la primera línea de código solo se escribe después de la última línea del diseño.

Un enfoque más económico.

Dado que el diseño lo realiza un equipo reducido de profesionales, en comparación con el equipo de desarrollo, es más barato diseñar que programar. Sumado a eso, evitar distorsiones y cambios genera ahorros importantes, comparado con la metodología de la lista de features.

Además de lo que implica en los costos directos del proyecto, el enfoque basado en el diseño de los proyectos es capaz de predecir con más exactitud la duración y recursos necesarios para un proyecto, por lo que los costos indirectos, derivados del costo de oportunidad que implica tener o no tener en determinada fecha el proyecto terminado serán mucho menores.

Por último, es más económico en dolores de cabeza. No se trata de un camino adornado con pétalos de rosa, pero una cosa es enfrentar los problemas y dificultades naturales de un proyecto complejo y otra cosa es tener un proyecto fuera de control, pasar meses o incluso años recriminando las culpas entre los distintos sectores y terminar en muchos casos en un ajuste de cuentas cuando sea necesario asumir las pérdidas.

Los programadores no deben diseñar

Todas las profesiones y oficios requieren de determinadas características personales. La programación no es ajena a esta regla. Para ser programador es necesario poseer un conjunto de atributos que no están presentes en la mayoría de los humanos: pensamiento lógico estricto, capacidad de determinar el modelo interno a partir del comportamiento exterior, comprensión inmediata de la casuística de un problema, incluyendo todos los casos límites y algunas otras.

Por ejemplo, la expresión “los clientes de Uruguay y Argentina”, completamente comprensible para la mayoría de los mortales, sufre de enormes carencias lógicas. Para empezar, si asumimos que un cliente vive en un solo país, la expresión correcta sería “los clientes de Uruguay o Argentina” dado que “los clientes de Uruguay y Argentina” es un conjunto vacío. Ahora faltaría determinar que pasa cuando un cliente vive en más de un país, lo que automáticamente determina si guardaremos una o más

Jamás viole esta ley: la primera línea de código solo se escribe después de la última línea del diseño.

direcciones para cada cliente, porque si aceptamos que un cliente puede vivir en más de un país, pero tenemos sólo una dirección, entonces "los clientes de Uruguay y Argentina" serán en nuestro sistema el conjunto vacío. Ahora viene el tema de si un cliente es de Uruguay cuando compra en Uruguay, cuando vive en Uruguay, cuando es de nacionalidad Uruguayo o cuando lo atiende un Representante de Ventas de Uruguay, cuando ocurren todas o una combinación de ellas. Después de eso llega el problema de las personas físicas versus las personas jurídicas. Y así hasta el infinito.

Si un programador es incapaz de pensar así, si esta forma de razonar no es una capacidad natural en su forma de ser, jamás será un buen programador. Sus programas estarán llenos de situaciones inconsistentes, en las que el software o se cuelga, o no produce el resultado deseado o produce alguna otra consecuencia indeseable.

Pero esa misma habilidad es la que la que lo inhabilita a diseñar. Sus necesidades y su forma de ver y concebir el sistema son diametralmente opuestas a las de quien va a usar el sistema. Donde el usuario ve "y" el programador verá "o". El diseño debe ser llevado adelante por profesionales que entiendan de programación, que merezcan el respeto del equipo de programación, pero que tienen capacidades y habilidades que éstos no poseen.

Para muestra basta un botón

Tres ejemplos reales y de Uruguay:

1 - Una empresa solicitó una cotización a tres proveedores con una especificación completa del sistema necesario para ejecutar una promoción de corta duración. Dado que el software iba a ser usado durante apenas unas semanas, se había eliminado deliberadamente la opción "eliminar participantes" y "modificar participantes" a los efectos de simplificar el funcionamiento. Los participantes sólo se darían de alta y si se cometían errores se darían de alta nuevamente, dado que esto no afectaba los resultados de la promoción. A pesar de ello, las tres cotizaciones incluyeron las opciones "eliminar participantes" y "modificar participantes". Cuando se les pregunto porqué a los oferentes, los tres dijeron que les había parecido una omisión involuntaria, y argumentaron que de todos modos, era una funcionalidad más, que no sería de ningún modo nociva para el usuario final.

El diseño debe ser llevado adelante por profesionales que entiendan de programación, que merezcan el respeto del equipo de programación, pero que tienen capacidades y habilidades que éstos no poseen.

2 - En las pruebas de una aplicación Web, en la que empresas clientes interactuaban con su proveedor, se detecto que al eliminar una empresa cliente aparecía el mensaje "el cliente tiene usuarios activos, debe eliminarlos antes de dar la baja". Los testeadores se sorprendieron, dado que la especificación completa del sistema no contenía el concepto de usuario, es más, la palabra no se mencionaba en ninguna parte del documento, ni en los diagramas de las pantallas del sistema, que habían sido especificadas una por una. Al ser consultados los programadores sobre el mensaje de error, argumentaron que el sistema internamente utilizaba el concepto de usuario para manejar la sesiones a través de la Web. El proveedor indicó que a pesar de eso quería que el software se ciñera estrictamente a la especificación, a lo que los programadores contestaron que esa implementación iba a permitir

que en un cliente hubiera varios usuarios distintos y dado que la funcionalidad ya estaba hecha, para qué eliminarla.

3 - La especificación de un software de manejo de eventos, suponía la administración de un evento por vez para cada usuario del sistema. Cuando los programadores mostraron por primera vez el sistema (que había sido diseñado en detalle y de forma adecuada) la pantalla de manejo de eventos en vez de ir al único evento tal como la especificación lo indicaba, mostraba una lista de eventos, donde se podían incluir todos los eventos que el usuario quisiera. Una vez más los programadores argumentaron que así el sistema tendría más posibilidades y que no veían sentido alguno en eliminarlo. A pesar de que el cliente definió el regreso al diseño original, la implementación interna de múltiples eventos se conservó en el interior del software (el código es como el hormigón!!!) y fue la causa de frecuentes inconsistencias y errores en el aplicativo.

Si una empresa invirtió el tiempo, esfuerzo y dinero necesario para diseñar adecuadamente un producto basado en software, no espera que quienes lo llevan adelante lo cambien. Sin embargo la lógica del programador, acostumbrado a comenzar a codificar sin diseño alguno, se siente en la libertad y en la obligación de incorporar todo aquello que crea conveniente. Que una funcionalidad se halla descartado porque aumentaría la necesidad de soporte al usuario, o complicaría el modelo conceptual, o cualquier otra causa, no es hoy un argumento convincente para el equipo de desarrollo.

No se desanime, no está solo

Dado que los problemas de los proyectos informáticos no son visibles en el comienzo, sino que se desarrollan in-crescendo a medida que el proyecto avanza, será para usted muy difícil convencer a sus pares de que se necesita una metodología completamente distinta, que debe ser aplicada desde el comienzo. A esto se suma para los gerentes no-informáticos la dificultad para moverse en el ámbito de las tecnologías de la información modernas, la utilización de un lenguaje críptico, sólo para iniciados y la interacción con una forma de pensar y razonar los problemas basada en la lógica matemática.

Si llegó hasta este último párrafo, es porque se sintió completamente identificado con la problemática que se plantea. No se desanime, converse sinceramente con sus pares, dentro y fuera de la empresa y comprobará rápidamente que sus problemas no son solo suyos, sino de toda la interacción con la industria informática de hoy en día. Provienen de un status quo asumido como válido y que debe ser modificado. Aplique todas sus fuerzas y recursos para cambiar ese status quo, adoptando metodologías basadas en un enfoque nuevo y los resultados no se harán esperar.

Artículos relacionados

Diseño, los usuarios al poder - Versión PDF 80KB
(http://www.mordecki.com/ebusiness/usabilidadI/usabilidad.PDF)

Los objetivos del diseño - Versión PDF 39KB
(http://www.mordecki.com/ebusiness/objetivos_del_diseno.pdf)