AQS

Advanced Quality Solutions

GESTIONAR PROYECTOS IT CON ÉXITO

Enero 2002

Copyright © 2001 Advanced Quality Solutions

ÍNDICE

ÍNDICE 1

SUMARIO 3

ERRORES CLÁSICOS EN LA GESTIÓN DE PROYECTOS TI 4

UN EJEMPLO: MICROSOFT WORD 1.0 4
ERRORES RELACIONADOS CON EL FACTOR HUMANO 5
ERRORES RELACIONADOS CON EL PROCESO 9
ERRORES RELACIONADOS CON EL PRODUCTO 11
ERRORES RELACIONADOS CON LA TECNOLOGÍA 12
CONCLUSIONES 13
RESUMEN DE LOS ERRORES CLÁSICOS 13

MOTIVACIÓN 14

FACTORES DE MOTIVACIÓN TÍPICOS EN DESARROLLADORES 14
LOS CINCO FACTORES DE MOTIVACIÓN MAS IMPORTANTES 16
LOGRAR OBJETIVOS 16
POSIBILIDAD DE DESARROLLO PROFESIONAL 17
EL CONTENIDO DEL TRABAJO 17
VIDA PRIVADA 18
OPORTUNIDAD DE SUPERVISIÓN TÉCNICA 18
FACTORES QUE “MATAN” LA MOTIVACIÓN 19
FACTORES DE “HIGIENE” 19
OTROS FACTORES IMPORTANTES 19

CICLO DE VIDA DEL PROYECTO 22

MODELO EN CASCADA 22
VENTAJAS 22
INCONVENIENTES 22
DESARROLLO EN ESPIRAL 23
VENTAJAS 23
INCONVENIENTES 24
DESARROLLO EVOLUTIVO ORIENTADO A PROTOTIPOS 24
VENTAJAS 24
INCONVENIENTES 24
EXTREME PROGRAMMING 25
VENTAJAS 27
INCONVENIENTES 27
DESARROLLO ORIENTADO A HITOS 28

VENTAJAS 28
INCONVENIENTES 28
MODELOS MIXTOS **29**

CONCLUSIONES **30**

BIBLIOGRAFÍA **31**

REFERENCIAS EN WEB **31**
LIBROS & ARTÍCULOS IMPRESOS **31**
LIBROS DE REFERENCIA 31
ARTÍCULOS Y FUENTES VARIAS 32

SUMARIO

Cuando se observan las tasas de productividad de las compañías de desarrollo de software se observa un curioso fenómeno: hay aproximadamente un margen de diferencia del 1 al 10, una tasa de diferencia difícil de encontrar en otras industrias. ¿A qué se debe esto?

En resumidas cuentas hay que buscar la respuesta en el hecho que la "industria" del software aún es joven, se puede decir que hasta hace relativamente poco era más bien una actividad artesanal. Aunque la importante evolución de la Ingeniería del Software ha aportado en los últimos años una nueva generación de metodologías y herramientas que hacen que podamos hablar por fin de una industria basada en la ingeniería hoy por hoy el fracaso de proyectos de tecnologías de la información sigue siendo un fenómeno demasiado frecuente, y es que el uso de una metodología adecuada y buenas herramientas son condición necesaria pero no suficiente para el éxito de un proyecto.

Deben ir acompañadas de una cultura de trabajo adecuada, esto quiere decir de una serie de prácticas y el saber hacer en las decisiones que se plantean ante los diversos problemas que surgen en la gestión de un proyecto.

Este artículo aporta en gran medida este saber hacer y ayudará así al lector a colocarse más cerca del "10". Para ello no pretende plantear ideas "revolucionarías", sino que se recopilan y sintetizan las conclusiones a las que han llegado múltiples autores basándose en lo que el sentido común les sugiere en base a sus experiencias en los proyectos que han vivido, es aquí de donde parte el artículo: del sentido común y la sensatez sobre una base de experiencia real.

De hecho una gran parte de las experiencias y conclusiones aquí expuestas le resultarán familiares al lector, quizás se anime así a poner en marcha algunas de las prácticas sugeridas en las que él posiblemente ya había pensado, pero no se sentía suficientemente respaldado para aplicarlas. En este sentido el público al que va dirigido este artículo es amplio e implica tanto a desarrolladores, como gestores como a los clientes, a todos este artículo les tiene algo que aportar, incluso para los profesionales de áreas cercanas a las tecnologías de la información como pueden ser la telecomunicaciones.

En definitiva se trata de evitar decisiones estratégicas y tácticas erróneas, poseer las cualidades humanas para conseguir un máximo bienestar en el equipo, y con ello una máxima motivación y productividad.

- ¿Hay que recuperar un proyecto fuera de plazo? ¡Añadamos más gente!
- ¿Hay cambios sustanciales en los requisitos del proyecto? Bueno, como estamos a la mitad del tiempo proyectado no es tan grave.
- Acaba de salir una nueva tecnología que promete reducir a la mitad del tiempo el desarrollo de tres módulos de los más importantes de nuestra aplicación, así que incorporémosla inmediatamente.
- …

¿Quien no ha vivido este tipo de situaciones y no ha sufrido sus consecuencias? Lo peligroso su carácter seductor ante las presiones típicas de un proyecto, por tanto, es importante hacer un ejercicio de concienciación y no dejarse seducir por "el lado oscuro" de la gestión de proyectos.

A lo largo del artículo se analizarán este tipo de "errores clásicos" y se discutirá cómo corregirlos. Por otra parte se dará especial importancia al factor más importante para la productividad de un proyecto de desarrollo: el factor humano, y por fin se examinarán también algunas de las principales estratégicas de cómo estructurar correctamente las actividades del ciclo de vida de un proyecto.

Copyright © 2001 Advanced Quality Solutions

ERRORES CLÁSICOS EN LA GESTIÓN DE PROYECTOS TI

Si bien disponer de un sólido conocimiento de lo que *no* se debe hacer en un proyecto supone un logro importante, hay que insistir en que no es suficiente con identificar los principales riesgos, que es lo que hace este documento a grandes rasgos, *hay que afrontarlos de forma proactiva*. Es decir, durante el desarrollo del proyecto hay que seguir monitorizándolos, analizarlos, priorizarlos y resolverlos.

Muchos de los puntos aquí listados parecerán evidentes, pero en la realidad se dan desgraciadamente con elevada frecuencia. En la mayoría de los casos esto se debe a la la subestimación de la magnitud de su impacto sobre el proyecto y factores psicológicos que hacen que las decisiones se toman más por lo que "pide el cuerpo" en un momento determinado que por razones objetivas. Para ver las magnitudes de las que estamos hablando, he aquí un ejemplo real:

UN EJEMPLO: MICROSOFT WORD 1.0[1]

El ejemplo de Microsoft Word 1.0 (la versión de MS-DOS, incomparable a la actual en complejidad) supone una buena lección en lo que refiere a las consecuencias de una mala planificación. Esta aplicación costó 5 años de desarrollo, consumiendo 660 hombres/mes[2] en esfuerzo y produjo un sistema de 249.000 líneas de código. WinWord tuvo una planificación extremadamente agresiva, la planificación del mejor caso para un proyecto de esta envergadura es de unos 460 días, la estimación para éste del caso más largo admitida era de 395 días, una diferencia de 65 días.

La productividad ha sido finalmente de 377 líneas de código por hombre/mes cuando una media de 1000-1500 líneas de buena calidad suele ser habitual. Esto da una idea del nivel de degradación que sufrió el proyecto.

¿Cómo fue posible tal desmesura? El desarrollo de WinWord cumplió con una serie de ejemplos típicos de cosas que pueden ir mal cuando un proyecto software se planifica demasiado agresivo:

- Winword sufría objetivos inalcanzables. La directiva de Bill Gates para el equipo fue "hacer el mejor procesador de texto de todos los tiempos" y hacerlo lo más rápido posible, preferiblemente en 12 meses, cualquiera de los dos objetivos era un desafío, los dos juntos eran sencillamente imposibles.
- Los tiempos agresivos impidieron una planificación precisa. En los cinco años se hicieron un total de 17 estimaciones de las cuales sola la segunda supero el año (se hizo nueve meses después de comenzar).
- El proyecto sufrió una rotación de personal extrema, tuvo 4 líderes distintos a lo largo de los cinco años, dos que abandonaron el proyecto por presión de tiempo excesiva y uno por razones de salud.
- Con la presión los desarrolladores descuidaron la calidad incluso de las partes críticas de la aplicación, diciendo que estaban "hechas", aun a sabiendas que no tenían la calidad necesaria y estaban incompletas. El resultado fueron 12 meses de "estabilización", cuando este período se estimaba en 3 meses.

[1] Fuente: *Rapid Development*. Microsoft Press 1996

[2] Es decir, un equipo con un media de 11 miembros

Copyright © 2001 Advanced Quality Solutions

El tiempo de desarrollo nominal (es decir, holgado) que se estimó después de un análisis a posteriori eran 26 meses, un tiempo de desarrollo "eficiente", 22 meses. Este tiempo hubiera sido sustancialmente más largo que el tiempo más pesimista planificado, pero hubiese dado el margen suficiente para evitar el grado de presión que degradó el proyecto por completo.

Es doloroso asumir un plazo de 22 meses para un proyecto que se quiere obtener en 12, pero forzar el plazo deseado no es la solución como este ejemplo demuestra contundentemente. **El plazo más corto es el que con mayor acierto se ha estimado.**

Ejemplos de compañías que han aprendido estas lecciones incluyen Boeing, la NASA, Microsoft, Ericsson, etc. Ellos utilizan planificaciones sistemáticas y realistas para evitar estos desastres y asumen los tiempos que los técnicos indican de cara a obtener un resultado determinado.

Ahora estamos en buenas condiciones para comprender la importancia de los siguientes párrafos:

ERRORES RELACIONADOS CON EL FACTOR HUMANO

- **Motivación insuficiente.** Estudio tras estudio se ha demostrado que el factor que más afecta a la productividad y calidad del desarrollo probablemente sea la motivación del personal. Resulta fácil degradarla con presiones por calendarios excesivas, y decisiones irracionales y por tanto hay que saber gestionarla. Este es un punto tan importante que le dedicamos un capítulo aparte, pero adelantemos aquí ya algunas ideas:

  Hay tres factores principales en la motivación de los desarrolladores:

  - *Las relaciones humanas y personal adecuado en los equipos.* El personal de la empresa no solamente se debe seleccionar por criterios puramente técnicos, sino también humanos. Resulta tan fundamental que los jefes de proyecto sean competentes en el ejercicio de su función como que tengan una personalidad adecuada. Por otra parte resulta muy recomendable buscar no solamente la cualificación técnica en los desarrolladores, sino intentar contratar también a "buena gente", un equipo que se lleva bien entre sí siempre será mucho más productivo que un equipo con un ambiente frío o incluso enemistado. El buen estar entre las personas del equipo conllevará casi como efecto secundario una alta productividad. Es fácil caer en la trampa de contratar a gente nueva por el plazo de su disponibilidad en vez de contratarla por la aportación al proyecto durante su tiempo de vida, hay que evitar este error a toda costa.
  - *La relación entre la empresa y su personal.* Decisiones aparentemente irracionales, promesas incumplidas y falta de comunicación son posiblemente los factores que más enturbian la relación de los empleados con la empresa. La empresa debe cuidar la comunicación con sus empleados y procurar que adquieran un poco de cultura de empresa. Si el equipo directivo logra de esta manera la confianza del personal se respetarán incluso aquellas decisiones que puedan parecer "irracionales", para ello no es necesario ni operativo explicar cada una de las decisiones estratégicas que toma la empresa, pero sí resulta recomendable escoger ocasiones puntuales para hacerlo, mantener sesiones periódicas de información sobre el estado de la empresa y conseguir que los desarrolladores se sensibilicen ante los problemas de gestión y negocio de la empresa. Cuanto más se sientan implicados los desarrolladores con la empresa, más se implicarán con sus proyectos.

Copyright © 2001 Advanced Quality Solutions

- *La filosofía de trabajo.* Varias estadísticas demuestran que la mayoría de los desarrolladores con cierta vocación ponen por encima de criterios económicos lo que les aporta su trabajo, hay dos facetas principales: A) Una forma de trabajo ordenada y participativa en la que cada uno de los miembros se sienta partícipe. B) El desarrollo personal de cada trabajador.
  Una gestión descontrolada del trabajo supone objetivos cambiantes constantes, estrés, falta de sensación de éxito y con ello frustración. Deteriora la moral de las personas y por tanto la calidad de su trabajo culminando en que tarde o temprano la persona se irá de la empresa. Esta situación se produce por una combinación de factores compleja y que se examinan uno a uno en los siguientes puntos del presente documento, limitémonos por ahora a decir, que una forma de trabajo desordenada es sencillamente, inadmisible.

  En cuanto al desarrollo personal resulta importante establecer una gestión en la que se den facilidades a cada empleado. Aquí no solamente cuentan las medidas costosas al alcance de las grandes compañías como planes de formación, etc. sino que pueden ser incluso mucho más eficaces medidas más "creativas" como "institucionalizar" simplemente un tiempo diario de investigación para cada desarrollador, la organización de eventos de intercambio de know-how entre los proyectos (workshops internos), animar a que se redacten artículos específicos sobre experiencias concretas en proyectos o nuevas tecnologías, tanto a nivel de gestión como tecnológico, serán una valiosa aportación a la gestión de conocimiento de la empresa y además se puede plantear la posibilidad de publicarlos, etc.

- **Empleados conflictivos.** No gestionar los problemas relacionados con personal problemático provoca también retrasos en el desarrollo de un proyecto. Es un problema común y se ha entendido bien desde que Gerald Weinberg publicó *Psychology of Computer Programming* en 1971. No actuar ante una situación así es la queja más frecuente de los miembros de un equipo sobre sus líderes (Larson y LaFasto 1989). La solución no tiene que pasar necesariamente por medidas radicales, sino que se debe gestionar con cierta delicadeza, muchas veces se debe a una determinada situación personal, falta de experiencia que lleva a malinterpretar la forma de actuar de su entorno, etc. Si se consiguen explorar las razones de fondo del comportamiento de la persona en cuestión muchas veces hay una solución que resultará positiva tanto para la(s) persona(s) como para la empresa y que incluso puede fortalecer los vínculos entre ambos.
- **Heroísmos.** Algunos desarrolladores piensan que acciones heroicas pueden aportar alguna clase de beneficio al proyecto (Bach 1995). Pero las experiencias demuestran que más bien son actitudes negativas para la gestión fiable de un proyecto puesto que problemas potenciales no se comunican hasta el último minuto distorsionando así el seguimiento del proyecto y se tiende a asumir riesgos demasiado altos. Este error es también muy seductor porque para muchos jefes de proyecto resulta tranquilizador que su equipo les transmita mensajes del tipo "lo podemos hacer", el jefe de proyecto se limita a creerlo y dormir más tranquilo, pero realmente está obviando la realidad y aplazando enfrentarse a una problemática que se agravará con el tiempo.

Copyright © 2001 Advanced Quality Solutions

- **Añadir personal a un proyecto retrasado.** Este es, sin duda, el fallo clásico más clásico. Y es que el factor humano no escala fácilmente, generalmente añadir gente a un equipo existente que se encuentra retrasado lo retrasa aún más puesto que genera mucho trabajo de coordinación adicional, además el equipo no cuenta en ese momento con la calma necesaria para dar a las personas nuevas la formación que necesitan del proyecto, con lo cual ésta irá lenta.
  Al final, la combinación de más canales de comunicación y necesidad de formación hace que los nuevos miembros del equipo le restan más tiempo de productividad al proyecto del que añaden y generan así retrasos aún mayores. Aunque de todas las maneras hay que hacer notar que esto depende evidentemente del perfil de cada proyecto, hay proyectos en los cuales sí resulta eficaz introducir más personal ante retrasos por requerir éste solamente una forma mínima o poder tener una estructura de módulos muy independientes. Lo que resulta importante es comprender es que añadir más gente a un proyecto no es ni mucho menos una fórmula mágica para resolver problemas de calendario, sino que hay una muy alta probabilidad de que sea todo lo contrario.
- **Oficinas ruidosas y faltas de espacio.** La mayoría de los desarrolladores valoran sus condiciones de trabajo como insatisfactorias, aproximadamente el 60% se queja además de que no hay suficiente silencio ni suficiente intimidad (DeMarco y Lister 1987). Trabajadores que desarrollan su actividad en oficinas silenciosas y con espacio tienden a ser más notablemente más eficientes que aquellos que trabajan en condiciones de ruido y falta de espacio.
- **Fricción entre desarrolladores y clientes.** Fricciones entre desarrolladores y clientes pueden surgir de varias maneras. Los clientes opinan que los desarrolladores no cooperan cuando rechazan el calendario exigido por ellos o cuando no entregan en los tiempos prometidos. Los desarrolladores piensan que los clientes insisten irracionalmente en calendarios irrealistas y cambios de requisitos imposibles después de que éstos hayan sido cerrados, las personalidades de ambos grupos pueden empeorar el conflicto aún más. La consecuencia principal es una comunicación pobre y por tanto una comprensión pobre de los requerimientos, junto con interfaces de usuario poco adecuados a las necesidades del cliente. Resulta difícil superar esta fricción por lo que hay que monitorizar este riesgo desde el principio detenidamente y emplear personal con "mano izquierda" que sepa resolver los conflictos y restaurar una comunicación eficaz.
- **Expectativas irrealistas.** Una de los causas más frecuentes de fricción entre desarrolladores y clientes o gestores son expectativas irrealistas. Sensibilizar al cliente no es un proceso trivial y requiere también una postura suficientemente razonable de su parte. Una vía de compromiso ante plazos que al cliente le parecen excesivos puede ser una cuidadosa negociación de los requisitos. Si se sabe negociar bien se encontrarán en la mayor parte de los casos requisitos que para el cliente resultan relativamente poco importantes pero sí suponen un ahorro de desarrollo que merece la pena. Otra vía puede ser una planificación orientada a prototipos o entregas sucesivas de módulos que adelanten funcionalidad importante aunque la versión completa se entregue en los plazos iniciales o incluso algo más tarde. Con este tipo de medidas posiblemente se puedan acercar posiciones en un principio alejadas.
- **Ausencia de apoyo efectivo de la dirección.** Todos los principios para la gestión exitosa de proyectos valen de poco si no son apoyados por la dirección de la empresa y ésta se deja llevar excesivamente por la presión de sus clientes forzando a los equipos a aceptar planificaciones irrealistas, etc. Según el consultor australiano Rob Thomsett esta falta de apoyo prácticamente garantiza el fracaso de un proyecto con cierto grado de ambición (Thomsett 1995).

Copyright © 2001 Advanced Quality Solutions

- **Compromiso insuficiente de los participantes en el proyecto.** Todos los participantes en un esfuerzo de desarrollo de software tienen que estar fuertemente comprometidos con el proyecto, tanto del lado del cliente como del lado del la empresa desarrolladora. Compromiso quiere decir fundamentalmente dedicar el tiempo y la profundidad necesaria a las tareas que correspondan a cada uno. Esto incluye tanto a los ejecutivos, como miembros del equipo de desarrollo, interlocutores del clientes, usuarios finales y cualquiera que se encuentre relacionado con el proyecto. Solamente de esta manera será posible una buena coordinación y capacidad de maniobra ante problemas.
- **Falta del "input" de usuario.** Resulta muy difícil para los no informáticos imaginar hasta qué punto se necesita el detalle de los procesos para implementarlos bien. TODOS los clientes tienden a quedarse muy en la superficie de los requerimientos, si los responsables del desarrollo lo admiten, esto se traduce normalmente en desarrollos que están lejos de cubrir las necesidades del cliente y con ello en retrasos para lograrlo, lo que a su vez provoca código "sucio" por los frecuentes cambios, mucho más difícil de mantener y evolucionar. Por otra parte resulta difícil para los desarrolladores imaginar hasta qué punto los usuarios se quedan en la superficie, ya que muchos requisitos que para ellos resultan evidentes los usuarios ni son capaces de imaginárselos y viceversa[3]. Debe tenerse en cuenta este dato: la Standish Group concluyó que la razón principal por la que los proyectos de software concluyen con éxito es por la participación comprometida de los usuarios.
- **Demasiados interlocutores o interlocutores ineficaces.** Normalmente se da este problema más de parte del cliente debido a que la participación en el proyecto para él supone una actividad con la que debe compaginar su trabajo diario, normalmente los responsables del lado del cliente prometen que las personas designadas se podrán alejar de su responsabilidad habitual para centrarse en el proyecto, pero en la realidad pocas veces se cumple, al final tienen que abordar todo su trabajo habitual más el trabajo del proyecto. El proyecto muchas veces se convierte en la segunda prioridad porque ante su empresa se les valora por su trabajo habitual, esto sencillamente resulta nefasto para cualquier proyecto. Otra faceta en la comunicación es el número de interlocutores, debería ser mínimo, con áreas de responsabilidad muy bien definidas y acotadas y con una dedicación plena. Aunque esto es difícil de conseguir resulta muy recomendable presionar al máximo al cliente para acercarse lo más posible a este ideal, el cliente lo agradecerá cuando el proyecto se haya realizado con éxito.

[3] Steve McConell pone un ejemplo muy ilustrativo de la distancia que separa los puntos de vista de usuarios y desarrolladores: propone que el lector se imagine un cliente experto en coches (pero no en ingeniería) que debe especificar la construcción de un coche a medida a un ingeniero que no es un especialista en la construcción de coches porque construye igualmente aviones, barcos, etc. El cliente hablará del motor, chasis, lunas, volante, etc. Pero imaginemos que se le olvidó especificar que al dar marcha atrás se deben encender unas luces blancas en la parte trasera del coche (algo evidente para él).

El ingeniero va hacer su trabajo y vuelve a los seis meses, cuando el experto ve el coche exclama "Vaya, me he olvidado especificar las luces de marcha atrás". El ingeniero se lleva las manos a la cabeza: "¿sabe usted lo que va a costar hacer esta modificación? ¡Tenemos que rediseñar la parte posterior del coche para incluir los faros nuevos, modificar la circuitería electrónica e incluir un sensor en el cambio! Esto llevará semanas como mínimo. ¿Porqué no me lo ha dicho antes?" El experto se resigna: "Parecía una petición tan sencilla..."

Copyright © 2001 Advanced Quality Solutions

- **Intereses "políticos".** Este punto se refiere tanto a conflictos de intereses entre los participantes en el proyecto, típicamente departamentos o responsables con malas relaciones, como miembros del proyectos cuyas pautas de actuación derivan más de la imagen hacia sus jefes y situación coyuntural que los intereses del proyecto o la empresa. Esta problemática resulta habitualmente invisible para los cargos más altos, debida a su falta general de tiempo que les hace recibir poca información la cual además fácilmente desvirtúa la realidad (o puede ser desvirtuada intencionadamente). Sin embargo, es interesante observar que la tasa de éxito de las estrategias "trepa" es considerablemente inferior de lo que comúnmente se piensa (Constantine 1995a).
- **"Wishful thinking".** Esto es básicamente el fenómeno de ponerse la venda en los ojos ante la voluntad que querer cumplir de cualquier manera un determinado objetivo por imposible que sea, ¡pero no hay que confundirlo con optimismo, el optimismo de por sí es sano para un proyecto! Cuantas veces se habrán escuchado frases como estas:
    - "Ninguno de los miembros del equipo realmente aceptada los plazos del proyecto, pero pensaron que trabajando duro y si las cosas salían bien, podrían ser capaces de sacarlo adelante."
    - "No ha hecho falta coordinar mucho los interfaces entre los módulos del producto, hemos estado hablando bastante y son relativamente simples, así que no debería haber muchos problemas en la integración."
    - "Al final hemos tenido que subcontratar un equipo de menor cualificación de lo especificado para el módulo de acceso a base de datos, pero son gente joven y muy motivados, seguro que lo sacan adelante a tiempo."
    - "No hace falta enseñar al cliente los últimos cambios al prototipo, ya le conocemos bien y le encantará como haremos la versión final."
    - "El equipo dice que va hacer un esfuerzo extraordinario para cumplir el plazo, se retrasaron en el primer hito el otro día, pero yo creo que a eso no hay que darle mucha importancia."
    - …

  Este fenómeno es muy frecuente y extremadamente peligroso, resulta frecuente tanto del lado del cliente que tiende a ignorar las valoraciones de los técnicos pensando que ya habrá una manera de hacer lo que necesita en el plazo que necesita como del lado de los desarrolladores que se ponen a trabajar duro para alcanzar "como sea" la fecha límite sin plantearse si es posible.
- **Confianza ciega en referencias establecidas.** Esto es básicamente el fenómeno de confiar hasta tal punto en grandes nombres que no se contrasta la cualificación que ofrecen. Es decir, contratar el equipo de una gran marca por si solo no implica en absoluto una garantía de calidad, hay que contrastar siempre la calidad del equipo que aportan.

ERRORES RELACIONADOS CON EL PROCESO

- **Planificación excesivamente optimista.** Una planificación excesivamente optimista impide una planificación eficiente y genera presiones que acaban degradando al equipo y proyecto en la mayoría de los casos. Una faceta posiblemente peor aún de una planificación excesivamente optimista es que no solamente no se llega a tiempo a los objetivos, sino que la dinámica en la que entra un proyecto en estas condiciones por la presión del tiempo hace que la productividad del trabajo sea sensiblemente menor que en un proyecto correctamente planificado.

Copyright © 2001 Advanced Quality Solutions

- **Gestión de riesgos insuficiente.** Resulta muy fácil llegar a un punto en el cual no queda capacidad de maniobra si se detecta que el proyecto va mal cuando no hay una gestión de riesgos activa. Este hecho es especialmente relevante si se tiene en cuenta que muchas veces un solo factor de los aquí expuestos puede degradar el proyecto por completo.
- **Fallos debidos a la subcontratación.** A veces se subcontratan partes de un proyecto que por presiones del calendario y falta de personal especializado. Pero las empresas subcontratadas con frecuencia entregan el trabajo tarde, con un nivel de calidad inaceptable o no conforme a los requisitos (Boehm 1989).
- **Abandonar la planificación ante la presión del tiempo.** El problema no es tanto el abandono de la planificación en sí como el hecho de que casi todos los equipos de desarrollo caen en la trampa psicológica de entrar en el "code-and-fix" (programar y arreglar a toda velocidad sin demasiada meditación) ante presiones de tiempo altas. El trabajo a partir de ese punto se convierte en descoordinado y más ineficiente que antes.
- **Pérdida de tiempo en la fase preparativa de un proyecto.** La fase preparativa de un proyecto es el tiempo que transcurre entre su aprobación y su presupuestado. No es raro que este proceso dure meses, a veces incluso años y luego plantee, sin embargo, una planificación de la ejecución del proyecto muy agresiva. Es mucho menos arriesgado acortar la fase preparativa y disponer de tiempo para la ejecución.
- **Plantear demasiados objetivos a la vez.** Este error es otro de los "muy frecuentes", tanto a nivel microscópico como macroscópico. El punto de vista microscópico se refiere al día a día de la gestión del proyecto, es decir, los objetivos a corto plazo que el jefe de proyecto va planteando al equipo. Plantear demasiados objetivos en paralelo es uno de los factores que en mayor medida impiden que un equipo trabaje de forma efectiva. El punto de vista macroscópico se refiere a los objetivos que se plantean para el producto, véase el punto "Plantear demasiados objetivos a la vez" de la sección de errores relacionados con el producto.
- **Cortar actividades fundamentales ante un retraso.** En muchos proyectos acciones de recorte de actividades han sido, por ejemplo, no dedicar el tiempo necesario a la especificación de requisitos, análisis y diseño. Recortar actividades fundamentales como éstas desde luego empeora con garantías aún más la situación.
- **Diseño inadecuado.** Este es un caso particular del punto anterior, pero especialmente delicado por el alto grado de creatividad y por tanto de cierta calma que requiere para ser realizado con éxito. Muchas veces la presión convierte las decisiones de diseño en decisiones oportunistas frente a las necesidades del momento que provocan la necesidad de uno o más rediseños del sistema antes de entregar el proyecto. Finalmente los rediseños suman más tiempo del que hubiese requerido un diseño bueno desde el principio.
- **Control de calidad insuficiente.** Otro punto típico de recorte es el control de calidad, es decir, eliminar actividades como la revisión del diseño y el código, planes de pruebas, etc. Estudios demuestran que recortar un día de actividades QA (Quality Assurance) en las fases tempranas de un proyecto es probable que cueste entre 3 a 10 días de actividad posterior (Jones1994). Es decir, al final se alarga el proyecto.

Copyright © 2001 Advanced Quality Solutions

- **Control de gestión insuficiente.** Es importante llevar a cabo un control de la evolución del desarrollo para ver si los hitos planteados se van cumpliendo y diagnosticar las causas de los posibles problemas puesto que esto permite reaccionar a tiempo y tomar decisiones razonables dentro de la situación del proyecto.
- **Convergencia prematura.** Poco antes del momento planificado para la entrega de un producto, hay una fase frenética para pulir la velocidad, la documentación final, incorporar ayuda en línea, rematar el programa de instalación, etc. En proyectos bajo presión existe la tendencia de entrar en esta fase demasiado pronto, lo que repercute en repetirla media docena o más de veces hasta que realmente se concluye.
- **Omitir tareas necesarias de las estimaciones.** En todos los proyectos hay un alto porcentaje de tareas y requisitos poco evidentes, pero necesarios. Aquí entran puntos como tiempos de respuesta del sistema, determinados aspectos ergonómicos del interfaz de usuario, procesos secundarios (como backups) derivados de los procesos especificados, etc. Su omisión se penaliza frecuentemente con desviaciones entre un 20% y un 30% sobre el tiempo total del proyecto.
- **Planificar para recuperar retrasos dentro de la planificación inicial.** Una manera muy frecuente de corregir estimaciones ante un retraso frente a un hito determinado es planificar una compensación posterior sin alterar los tiempos globales del proyecto. Es decir, cuando sobre un proyecto de 6 meses se llega al hito del mes 2 en el mes 3 es habitual decir "ya lo recuperaremos en lo que queda de proyecto, tenemos aún 3 meses por delante y ya sabemos más del proyecto". Aunque es cierto que la productividad de por sí crece conforme se conoce mejor al producto desarrollado, la mayoría de los proyectos no recuperan nunca de esta manera sus retrasos iniciales.
- **Programación "code-like-hell".** Algunas compañías creen que una codificación rápida con gente motivada es lo que hace falta realmente para finalizar los proyectos en el menor tiempo y que de esa manera se supera cualquier obstáculo. Los riesgos expuestos en este capítulo y sus razonamientos deberían dejar claro que están muy equivocados. Recientemente han aparecido propuestas como el "Extreme Programming (XP)" que no se deben confundir con en esta filosofía.

ERRORES RELACIONADOS CON EL PRODUCTO

- **Exceso en los requerimientos.** Muchos proyectos tienden a incluir más requerimientos de los que realmente necesitan, requisitos complejos añaden tiempos de desarrollo desproporcionados con respecto a los requisitos fundamentales. Luego hay analizar muy bien cuales quiere realmente el cliente para su negocio y priorizarlos correctamente.
- **Plantear demasiados objetivos a la vez.** Relacionado con el producto se refiere al nivel macroscópico, es decir, a los objetivos globales del proyecto del proyecto. Pedir que un proyecto justo en plazos sea además óptimo en el uso de la memoria, óptimo en velocidad de respuesta, óptimo en facilidad de uso y tenga una legibilidad del código perfecta será un conjunto de objetivos prácticamente imposible de alcanzar. Asumir un consumo relativamente elevado de memoria (es un recurso muy barato hoy en día) a cambio de una máxima velocidad de respuesta, junto con una facilidad de uso buena (si el perfil de usuarios lo permite) y una legibilidad de código aceptable puede ser una sabia rectificación en virtud del éxito del proyecto y producirá una calidad del resultado prácticamente similar.

Copyright © 2001 Advanced Quality Solutions

- **Requerimientos inestables.** El enemigo más feroz de la calidad del producto es el constante cambio de los requerimientos puesto que hacen sencillamente imposible un buen diseño, de hecho es uno de los factores principales en el fracaso de los proyectos. Es muy importante concienciar al usuario de esto, ha de tenerse en cuenta que la informática es una ingeniería de tecnología puntera, con un alto grado de dificultad que requiere diseños cuidadosos.
- **Desarrollo no centrado en los objetivos del proyecto por parte de miembros del equipo de desarrollo.** Muchos programadores tienden a fascinarse con los nuevos avances tecnológicos y con la implementación de "virguerías", tanto por disfrute personal como por la ambición de superar ciertos retos. Esta tendencia tiene que ser controlada porque puede degenerar en una pérdida de tiempo importante, aunque también se debe respetar algo de libertad para mantener la motivación a un buen nivel. Aquí el criterio personal del jefe de proyecto es la clave.
- **Negociación "tira y afloja".** Un tipo de negociación estrafalaria sucede cuando un responsable que aprueba un retraso en la planificación de un proyecto que transcurre más lento de lo esperado añade a la vez tareas completamente nuevas después de la aprobación de los nuevos plazos. Muchas veces hay motivos psicológicos de presión ante superiores que hacen que el responsable se sienta forzado a "compensar" el retraso con más funcionalidad (muchas veces poco útil, además), pero este alivio a corto plazo es traidor porque realmente contribuirá a alargar el proyecto aún más y el enfrentamiento con sus superiores será peor.
- **Desarrollo orientado a investigación.** El estar en la cresta de la tecnología convierte la planificación en pura especulación, hay que encontrar el punto intermedio óptimo entre los beneficios de las nuevas tecnologías y las áreas de aplicación de éstas en el desarrollo. En este sentido un apoyo I+D externo al proyecto es de gran utilidad, ya que con su ayuda se podrá calibrar la fiabilidad y potencial de mejora de la nueva tecnología y decidir así con criterio si la relación entre beneficio y riesgo hace recomendable utilizarla o no en un determinado proyecto.

ERRORES RELACIONADOS CON LA TECNOLOGÍA

- **El síndrome de la panacea.** La creencia ciega en tecnología y herramientas que no han sido probadas y se consideran la solución mágica a todos los problemas, la tendencia a esto es elevada en muchas personas.
- **Ahorros sobreestimadas por herramientas o métodos nuevos.** Los beneficios de nuevas herramientas o métodos de trabajo se ven anulados a corto/medio plazo por la curva de aprendizaje que supone su dominio correcto. Eso hace difícil que dentro de un proyecto se consigan resultados espectaculares, sobre todo, teniendo en cuenta que el tiempo disponible para aprendizaje se tiende a minimizar. Por otra parte el uso de nuevas herramientas o métodos implica nuevos riesgos que solamente se descubren con el tiempo.
- **Cambio de herramientas en medio del proyecto.** Resulta muy arriesgado cambiar la herramientas de desarrollo durante el proyecto puesto que los fallos en su uso y una curva de aprendizaje más costosa de lo esperado pueden anular fácilmente lo beneficios esperados e incluso descompensarlos. El entorno tecnológico ha de ser escogido con calma antes del proyecto y ser mantenido durante su realización.

Copyright © 2001 Advanced Quality Solutions

- **Ausencia de sistemas de control de versiones de código fuente.** Esto es básico, pero desgraciadamente resulta increíblemente frecuente que se destruyan partes importantes de código y recursos por la ausencia de sistemas de control de versiones y backup.

CONCLUSIONES

Desarrollar productos software es una tarea complicada. Concebir un sistema de información complejo ya es de por sí un reto, si unimos este hecho a que por lo general la tecnología empleada es muy sofisticada (y más hoy en día) queda claro que solamente se puede afrontar con éxito si se gestiona con cuidado y los medios de ingeniería adecuados.

En los párrafos anteriores hemos relacionado los errores más comunes y más importantes, pero hay muchos más. Resulta muy recomendable actualizar esta lista continuamente con los errores que se detecten en cada uno de los proyectos abordados.

Resumen de los errores clásicos

Factor humano	Proceso	Producto	Tecnología
1. Motivación insuficiente	1. Planificación excesivamente optimista	1. Exceso en los requerimientos	1. El síndrome de la panacea
2. Empleados conflictivos	2. Gestión de riesgos insuficiente	2. Plantear demasiados objetivos a la vez	2. Ahorros sobreestimados por herramientas o métodos nuevos
3. Heroísmos	3. Fallos debidos a la subcontratación	3. Requerimientos inestables	3. Cambio de herramientas en medio del proyecto
4. Añadir personal a un proyecto retrasado	4. Abandonar la planificación ante la presión del tiempo	4. Desarrollo no centrado en los objetivos del proyecto por parte de miembros del equipo de desarrollo	4. Ausencia de sistemas de control de versiones de código fuente
5. Oficinas ruidosas y faltas de espacio	5. Pérdida de tiempo en la fase preparativa de un proyecto	5. Negociación “tira y afloja”	
6. Fricción entre desarrolladores y clientes	6. Cortar actividades fundamentales ante un retraso	6. Desarrollo orientado a investigación	
7. Expectativas irrealistas	7. Plantear demasiados objetivos a la vez		
8. Ausencia de apoyo efectivo de la dirección	8. Diseño inadecuado		
9. Compromiso insuficiente de los participantes en el proyecto	9. Control de calidad insuficiente		
10. Falta de “input” de usuario	10. Control de gestión insuficiente		
11. Demasiados interlocutores o interlocutores ineficaces	11. Convergencia prematura		
12. Intereses “políticos”	12. Omitir tareas necesarias de las estimaciones		
13. “Wishful thinking”	13. Planificar para recuperar retrasos dentro de la planificación inicial		
14. Confianza ciega en referencias establecidas	14. Programación “code-like-hell”		

Copyright © 2001 Advanced Quality Solutions

AQS

MOTIVACIÓN

Dentro de las áreas que hemos identificado anteriormente con relación a los errores que se pueden cometer en un proyecto de desarrollo, el factor humano tiene el mayor potencial para acelerar un proyecto si se gestiona adecuadamente. Gestionar aquí no significa presionar, sino encontrar los factores que hagan que los desarrolladores alcancen su máximo nivel de productividad y que lo hagan a gusto, en definitiva: encontrar la forma de motivarlos al máximo y mantener ese alto nivel de motivación de una forma continuada.

La motivación es sin lugar a dudas el factor más importante con diferencia en cuanto a la productividad de las personas, son muchos los estudios que avalan esta afirmación (Boehm 1981).

La trascendencia de este factor queda clara desde la siguiente perspectiva: Varios investigadores han identificado en sus estudios diferencias de productividad entre desarrolladores con la misma experiencia en órdenes de magnitud entre 1 y 10, e incluso mayores en casos determinados (Sackman, Erikson, y Grant 1968; Curtis 1981; Mills 1983; DeMarco y Lister 1985; Curtis et al. 1986; Card 1987; Valet y McGarry 1989).

Aunque estas cifras pueden parecer extremas cualquiera con algo de experiencia en la industria del software sabe que las diferencias entre desarrolladores malos, mediocres y brillantes son muy elevadas. No se producen tanto por diferencias en inteligencia o talento (aunque estos son indudablemente dos factores de mucho peso), sino mucho más por la actitud de trabajo: mantenerse al día o no con la evolución tecnológica, conformarse con la primera solución que se encuentra ante un problema o tener la voluntad de investigar un problema a fondo y encontrar la solución más adecuada, diseñar pensando en resolver exclusivamente los problemas del corto plazo o proyectar una solución teniendo en mente la problemática futura de las siguientes etapas del proyecto, etc., etc., etc. en definitiva: trabajar con ganas o no.

Es precisamente el potencial de "efecto palanca" que las soluciones tecnológicas óptimas tienen sobre los tiempos de desarrollo el que marca estas diferencias tan pronunciadas entre la productividad de los desarrolladores, por eso es clave conseguir que los desarrolladores tengan la voluntad de conseguir resultados óptimos y para ello deben estar motivados. Ni falta hace decir que la desmesura en este punto tampoco es buena, volveríamos al error que identificamos con respecto a la tendencia de crear "virguerías" y no centrarse realmente en los objetivos del proyecto.

FACTORES DE MOTIVACIÓN TÍPICOS EN DESARROLLADORES

Para encontrar la manera de motivar a un determinado tipo de personalidad es importante hacerse primero una idea de sus rasgos más relevantes, en este sentido son interesantes datos como los obtenidos en el test "Myers-Briggs Type Indicator" (MBTI). Este test divide la personalidad de un profesional en cuatro dimensiones:

- Extrovertido (E) o introvertido (I)
- Sensitivo (S) o intuitivo (N)
- Pensativo (T) o impulsivo (F)
- Juzgador (J) o perceptivo (P)

De aquí salen 16 combinaciones de cuatro letras, lo que significa que se consideran 16 tipos de personalidad.

Copyright © 2001 Advanced Quality Solutions

Resulta curioso con qué claridad se identifica una personalidad "típica" de desarrollador, la cual se caracteriza por los siguientes rasgos:

- Aproximadamente el 50-66% de los desarrolladores son introvertidos frente a un 25-33% de la población general. Es decir los desarrolladores tienden en general hacia su interior, el mundo de las ideas, su desarrollo personal, dan menos importancia a cuestiones exteriores como la imagen, status, etc.
- El 80% de los profesionales informáticos tienden a un perfil reflexivo frente al 50% de la población general. Es decir, tienen una clara preferencia hacia la toma de decisiones basadas en la lógica y razonamientos claros y no tanto en valores subjetivos o emocionales.
- En consonancia con este hecho está la preferencia por juzgar (66%) frente a percibir, es decir, los desarrolladores tienden a vivir de una manera ordenada y planificada cuando los perfiles perceptivos tienen a adaptarse más a las circunstancias de cada momento.

Para ver un ejemplo de cómo se deben tener en cuenta estos rasgos de personalidad basta con plantear la forma de actuar ante la frecuente situación de retos ambiciosos en un proyecto: más vale utilizar argumentos lógicos y creíbles para motivar a los desarrolladores que utilizar fórmulas emotivas, sin argumentos convincentes. Raramente un grupo de desarrolladores responderá positivamente a objetivos imposibles.

A continuación aportamos una tabla con unos datos más detallados[4], aunque estos datos ya tienen más de 20 años y dependen también de circunstancias coyunturales como la situación económica, etc., siguen siendo muy válidos y merece la pena tenerlos en cuenta a la hora de gestionar equipos de desarrollo.

Concluyendo podemos afirmar que:

- *Comparado con la población en general,* a los desarrolladores les motiva *mucho* más la posibilidad de desarrollo personal, vida privada, oportunidad de supervisión técnica, y relación personal con sus compañeros. Les motivan menos factores de "status", relación personal con subordinados, responsabilidad y reconocimiento.
- *Comparados con sus gestores,* a los desarrolladores les motiva *algo* más la posibilidad de desarrollo personal, vida privada, y oportunidades de supervisión técnica. Les motivan menos factores de "status", relación personal con subordinados, responsabilidad y reconocimiento.

Esta última comparación entre desarrolladores gestores y técnicos resulta especialmente interesante ya que indica que los gestores **no** deben motivar a su equipo con los criterios que aplicarían a si mismos. El hecho de la responsabilidad en si misma que para un gestor es el factor fundamental, para un desarrollador tiene poco peso frente la posibilidad de enfrentarse a retos técnicos, evolucionar en conocimientos, tomar su propias decisiones dentro de su área en su trabajo, etc.

[4] Fuente: Software Engineering Economics (Boehm 1981) y "Who is the DP Profesional?" (Fitz-enz 1978).

Copyright © 2001 Advanced Quality Solutions

Veamos ahora los datos en detalle:

Analista programador	Jefe de proyecto	Población general
1. Lograr objetivos	1. Responsabilidad	1. Lograr objetivos
2. Posibilidad de desarrollo profesional	2. Realización personal	2. Reconocimiento
3. El contenido del trabajo	3. El contenido del trabajo	3. El contenido del trabajo
4. Vida privada	4. Reconocimiento	4. Responsabilidad
5. Oportunidad de supervisión técnica	5. Posibilidad de desarrollo profesional	5. Progreso
6. Progreso	6. Relaciones interpersonales con subordinados	6. Salario
7. Relaciones interpersonales con compañeros	7. Relaciones interpersonales con compañeros	7. Posibilidad de desarrollo personal
8. Reconocimiento	8. Progreso	8. Relaciones interpersonales con subordinados
9. Salario	9. Salario	9. Status
10. Responsabilidad	10. Relaciones interpersonales con superiores	10. Relaciones interpersonales con superiores
11. Relaciones interpersonales con superiores	11. Políticas corporativas y de administración	11. Relaciones interpersonales con compañeros
12. Seguridad laboral	12. Seguridad laboral	12. Oportunidad de supervisión técnica
13. Relaciones interpersonales con subordinados	13. Oportunidad de supervisión técnica	13. Políticas corporativas y de administración
14. Políticas corporativas y de administración	14. Status	14. Condiciones de trabajo
15. Condiciones de trabajo	15. Vida privada	15. Vida privada
16. Status	16. Condiciones de trabajo	16. Seguridad laboral

LOS CINCO FACTORES DE MOTIVACIÓN MAS IMPORTANTES

Trabajar en los cinco factores principales de motivación puede ser de gran ayuda para lograr acercarse más al deseable factor 10 de rendimiento, se trata de crear un ambiente de trabajo en el cual los desarrolladores puedan realizar sus inquietudes. Cuando las personas disfrutan, ellos mismos se comprometerán y responsabilizarán de cumplir los objetivos de trabajo, y si es necesario echar horas extra, lo harán sin mayores inconvenientes.

Lograr objetivos

Los desarrolladores de software disfrutan generalmente de su profesión. Por tanto, la mejor motivación para ellos es proporcionarles un entorno de trabajo que les permita hacer lo que más les gusta: desarrollar software.

Un pilar fundamental para que los desarrolladores se encuentren motivados para lograr sus objetivos es hacer que se sientan plenamente partícipes del proyecto, para ello resulta fundamental que desde el principio se impliquen en las decisiones como lo son la depuración de los objetivos y la planificación de los plazos. Que los desarrolladores participen en estas actividades hace que los resultados sean sensiblemente más realistas, al fin y al cabo hay que asumir que ellos tendrán en la mayoría de los casos un mejor criterio que los gestores para estimar los esfuerzos que les suponen unas tareas concretas.

El efecto psicológico del hecho de participar de esta manera es enormemente beneficioso para su motivación, son ellos quienes se comprometen a lograr los plazos y objetivos con lo cual se sentirán de verdad partícipes y comprometidos con el proyecto, será realmente "su" proyecto y saldrá de ellos hacer todo lo que esté en su poder para cumplir con su "palabra".

Copyright © 2001 Advanced Quality Solutions

Un argumento en contra de esta estrategia puede ser que los desarrolladores tenderán a aprovechar su participación para lograr unos plazos y objetivos lo más cómodos posibles, pero afortunadamente la experiencia demuestra que esto no es así, algo que resulta lógico si recordamos que el perfil típico del profesional del desarrollo es un perfil al que le gusta su trabajo y alcanzar metas ambiciosas. De hecho, los desarrolladores tienden más bien a estimar plazos y objetivos excesivamente ambiciosos, un buen jefe de proyecto deberá incluso tomarlos en este sentido con cierta cautela.

Los objetivos que se plantean deben ser claros y no excesivos en número, está demostrado que cuando se definen claramente los principales objetivos/prioridades de un proyecto y estos no son excesivos en el número, hay muy buenas posibilidades de lograrlos. La cuestión no está tanto en el número en sí de objetivos, aunque evidentemente también debe tender a ser pequeño, sino en que estos se puedan compaginar, véanse los puntos "Plantear demasiados objetivos a la vez" en el capítulo anterior.

Posibilidad de desarrollo profesional

Posiblemente la faceta más atractiva del desarrollo de software es el gran dinamismo del campo, es decir, hay que aprender y avanzar constantemente para mantenerse al día, no es muy sorprendente que las personas que hayan decidido trabajar en una profesión de estas características pongan peso en la faceta de las posibilidades de desarrollo profesional que les ofrece su trabajo.

Algunas medidas oportunas pueden ser las siguientes:

- Incentivar y remunerar los cursos internos
- Conceder tiempo o jornadas laborales flexibles para asistir a clases o estudiar
- Dedicar presupuesto a la creación de una buena biblioteca técnica
- Asignar los desarrolladores a proyectos que amplíen sus conocimientos técnicos
- Asignar a desarrolladores individuales o a grupos tareas de I+D para fines concretos (probar una nueva tecnología para la encriptación de datos, selección de un nuevo entorno de desarrollo, etc.)
- Asignar mentores a cada nuevo desarrollador (lo cual demuestra a ambos que la compañía se compromete con el desarrollo profesional de sus empleados)
- Evitar excesiva presión de calendarios para que haya un mínimo tiempo que se pueda dedicar a actividades como las mencionadas anteriormente
- Mantener una buena comunicación para monitorizar la consecución de los objetivos profesionales de cada desarrollador

El contenido del trabajo

Según Richard Hackman y Greg Oldham (Hackman y Oldman 1980), la motivación de los profesionales procede de tres fuentes principales:

- Que encuentren sentido en el trabajo que están realizando
- Deben sentir responsabilidad sobre los resultados de su trabajo
- Deben conocer los resultados reales de sus actividades

Hackman y Oldham identificaron cinco factores en el trabajo que contribuyen a estas fuentes de motivación:

- *Variedad de conocimientos aplicados.* Se refiere a la variedad de conocimiento que han de aplicarse en la ejecución de una tarea. Las personas encuentran más sentido en las tareas que exigen una variedad de conocimientos más amplia, incluso en tareas que no son muy importantes. Además el trabajo resulta más divertido de esta manera.
- *Identidad de la tarea.* Se refiere hasta qué punto la tarea requiere abordar un trabajo por completo. Las personas se responsabilizan más de su trabajo cuando perciben el trabajo como suyo o no se sienten solamente contribuidores a un trabajo en común.
- *Importancia de la tarea.* Es importante que el trabajo realizado se perciba como la creación de valor, como dicen Hackman y Oldham: los trabajadores que aprietan tuercas en un Boeing perciben su trabajo más importante que los que aprietan tuercas en un escaparate comercial.
- *Autonomía.* Este aspecto se refiere al grado de control del que se disfruta al realizar el trabajo, la sensación de ser su propio jefe dentro de su área de responsabilidad. Cuanto mayor sea esta sensación más responsabilidad se asume en el trabajo.
- *Realimentación del trabajo.* La realimentación del trabajo realizado es otro factor de motivación importante ya que aporta información directa de la efectividad de la persona que lo ha realizado. Es importante resaltar que no se trata tanto de los comentarios recibidos de un supervisor, sino lo que el trabajador ve por si mismo, el caso de un desarrollador de software es especialmente claro, ya que los programas creados proveen una realimentación inmediata cuando se ejecutan.

Vida privada

El respeto hacia la vida privada, es decir, el impacto de las jornadas laborales y el nivel de estrés sobre ésta tienen un peso muy alto para los desarrolladores. Cabe resaltar especialmente en este punto la diferencia de prioridades entre desarrolladores y jefes de proyectos, la vida privada ocupa el cuarto lugar para los primeros cuando para los últimos ocupa el puesto 15, esto puede provocar ciertos fallos en la gestión de un equipo, por ejemplo: ante una situación continuada de exceso de trabajo un jefe de proyecto puede pensar que los desarrolladores le darán poca importancia porque para él la tiene, pero se confunde al suponer que miden las cosas con su mismo criterio.

Lo mismo ocurre con la responsabilidad, ocupa el primer puesto para un jefe de proyecto, pero es de relativamente poca importancia para un desarrollador, por tanto asignar una alta responsabilidad a un desarrollador le puede parecer al jede de proyecto un "premio" que concede al desarrollador cuando para este supone quizás en un momento determinado más bien un factor de agobio por restarle tiempo de su vida privada.

Otra conclusión interesante de este hecho es que como medio de compensar las jornadas de trabajo en exceso puede ser más interesante conceder vacaciones extra que compensaciones económicas.

Oportunidad de supervisión técnica

Tener la oportunidad de realizar una supervisión técnica desde el punto de vista de un desarrollador supone un avance en el tipo de objetivos que debe lograr ya que ofrece la oportunidad de que sus decisiones técnicas trasciendan a un nivel mayor que el de sus responsabilidades personales en sus propias tareas. Sin embargo, para un jefe de proyecto supone casi un paso atrás porque esta acostumbrado a una supervisión de mayor nivel y no tener que preocuparse de los detalles técnicos, una supervisión técnica supone generalmente reducir su ámbito de acción y con ello la importancia de la tarea que percibe.

Copyright © 2001 Advanced Quality Solutions

FACTORES QUE “MATAN” LA MOTIVACIÓN

No solamente es importante saber cómo motivar, sino igualmente como no desmotivar. En ese sentido debemos hacer obligatoriamente una reflexión sobre las prácticas que debemos evitar en nuestra política de gestión de personal.

Factores de “higiene”

Los factores de higiene se refieren a aquellas condiciones “ambientales” que un trabajador necesita para poder realizar su trabajo de forma efectiva, su ausencia degrada considerablemente la motivación. Para los desarrolladores de software (y en general) estos son los principales factores:

- Iluminación y temperatura adecuada
- Suficiente espacio de mesa y armarios
- Suficiente silencio para permitir una buena concentración (incluida la posibilidad de apagar teléfonos)
- Suficiente intimidad para evitar interrupciones indeseadas
- Fácil acceso al equipo de oficina (fotocopiadora, fax, etc.)
- Disponibilidad de material de oficina (papel, bolígrafos, etc.)
- Acceso sin restricciones al ordenador
- Equipo informático razonablemente actualizado
- Buen soporte ante averías en el equipo informático
- Buena calidad de comunicaciones (teléfonos, email, Internet, ...)
- Disponibilidad de las herramientas software necesarias
- Disponibilidad del hardware necesario (por ejemplo, una impresora de color para un proyecto en el cual sea importante la impresión en color)
- Disponibilidad de manuales de referencia y otras publicaciones relevantes
- Un soporte mínimo de formación para nuevas herramientas, metodologías, etc.
- Uso de copias legales de software
- Flexibilidad horaria razonable (es decir, no se trata de condiciones anárquicas, sino márgenes razonables como entrar entre las 8:00 y las 10:00 horas, o que no haya impedimentos para tomarse ocasionalmente una tarde libre que se compensará en otra ocasión, etc.)

Otros factores importantes

- *Manipulación de parte de la dirección.* A los desarrolladores les gustan las cosas claras, como se ha visto anteriormente, son especialmente alérgicos a intentos de engaño (dramatizar en exceso la importancia de una fecha límite, intentar “convencer” a los desarrolladores de la viabilidad de un plazo claramente inviable, etc.) o presión irracional con calendarios imposibles. Si no se transmiten razones claras y no hay una flexibilidad de negociación mínima (para modificar algunos requisitos por ejemplo) los desarrolladores percibirán la sensación de que estas actuaciones como “por que sí”. La cuestión no es tanto si se trata de un intento de engaño real o no, sino lo que los desarrolladores perciben. Si no hay una mínima comunicación con ellos, es fácil que sientan una sensación de engaño incluso cuando hay razones sólidas para las decisiones tomadas.

Copyright © 2001 Advanced Quality Solutions

AQS

- *Presiones excesivas de calendario.* Una de las maneras más efectivas de afrontar un proyecto con una motivación nula es partir desde el principio de fechas límite imposibles. Muy poca gente se motivará ante una situación semejante a trabajar duro para sacar el proyecto adelante, para estar motivado hace falta una fe mínima en los objetivos. En el caso del perfil de un desarrollador de nuevo la faceta de alta racionalidad es la clave: una persona con un fuerte talante racional será incapaz de creer en la posibilidad de conseguir los objetivos y por tanto, se desmotiva.
- *Presiones innecesarias.* Una forma especialmente nefasta de gestión ocurre cuando los desarrolladores descubren en el desarrollo de un proyecto con mucha presión que ésta no proviene tanto del cliente, sino que ha sido la propia empresa quien ha generado la presión en un intento de conseguir una venta rápida y que el cliente hubiese estado dispuesto a esperar más tiempo a pagar más dinero ante la justificación adecuada de estos esfuerzos.
- *Falta de apreciación del esfuerzo de los desarrolladores.* Un fenómeno frecuente es la falta de apreciación ante los esfuerzos de los desarrolladores en un proyecto, esto se debe al hecho de que tanto para los clientes como para los gestores internos de alto nivel es difícil ver cuanto trabajo se está realizando realmente. Muchas veces se juzga desde fuera que determinada funcionalidad es fácil cuando en realidad no es así y se insinúa que los desarrolladores están trabajando poco o son poco productivos. Pero recordemos que los profesionales del desarrollo de software son gente muy con mucha motivación propia a la que les gusta trabajar en lo suyo, por tanto, acusarles o insinuarles que no se esfuerzan les resultará especialmente desmoralizante.
- *Dirección técnica incompetente.* Desarrolladores pueden ser motivados por jefes de proyecto técnicamente incompetentes siempre que estos asuman sus limitaciones técnicas y limiten sus decisiones a las no puramente técnicas. Pero un jefe que imponga de forma incompetente decisiones técnicas se convertirá en la burla del equipo. Un equipo no puede ser motivado por alguien a quien no respeta.
- *No implicar a los desarrolladores en decisiones que les afectan.* Un factor especialmente negativo, tanto para la moral de los desarrolladores como para el funcionamiento de la empresa es no implicar a estos en decisiones que les afectan. Esto se da con frecuencia, muchas veces simplemente por motivos de la organización de la empresa y no es solamente un factor de desmotivación, sino también una forma absurda de empeorar la eficacia de la empresa ya que con frecuencia son los desarrolladores quienes poseen los mejores criterios para estas decisiones. He aquí una lista de cuestiones en las que siempre se deberían implicar a los desarrolladores si se quiere mantener alta su moral:
    - Compromisos y ajustes de calendarios
    - Compromisos sobre nuevas funcionalidades, modificaciones o recortes.
    - Contratación de nuevos miembros para un equipo
    - Diseño del producto
    - Búsqueda de soluciones de compromiso ante plazos demasiado elevados, retrasos, etc.
    - Cambio o reestructuración de oficinas
    - Cambios en las herramientas utilizadas, tanto hardware como software
    - Compromisos imprevistos a lo largo de la ejecución de un proyecto, por ejemplo, un versión "prerelease" del producto o un prototipo reducido
    - Cambios organizativos y metodológicos

Copyright © 2001 Advanced Quality Solutions

- *Barreras a la productividad.* Un aspecto particularmente positivo del perfil de desarrollador son sus ganas de trabajar, algo envidiable para los responsables de muchas otras profesiones. Por eso resultan aún más flagrantes aquellos casos en los cuales los desarrolladores trabajan en entornos en los cuales se encuentran con constantes trabas para alcanzar una alta productividad, en estos casos es seguro que su motivación bajará sustancialmente.
- *Baja calidad.* Como personas a las que les gusta trabajar y con buenas dosis de creatividad una gran parte de su autoestima profesional deriva de la calidad de sus creaciones, por tanto impedir que los desarrolladores puedan hacer buenos productos es otro factor muy desmoralizante. Evidentemente no se pueden ignorar los condicionantes de la realidad del desarrollo de proyectos, pero como siempre se trata de una cuestión de mano izquierda para tener la sensibilidad de gestionar lo mejor posible esta problemática en un proyecto. Así, si este problema se plantea ante problemas de plazos un jefe de proyecto no debería forzar a los desarrolladores a implementar todas las funcionalidades "con alfileres", una alternativa adecuada puede ser buscar con ellos la manera de simplificarlas para que se puedan hacer con un aceptable nivel de calidad y defender estas propuestas con habilidad ante cliente.
- *Campañas de motivación mal enfocadas.* Cuidado con las campañas de motivación que insulten la inteligencia de los desarrolladores, una vez más hay que insistir en que son muy susceptibles a la falta de lógica y no responden bien a estrategias de motivación puramente emocionales. Es más efectivo un talante más moderado y realista.

Copyright © 2001 Advanced Quality Solutions

CICLO DE VIDA DEL PROYECTO

La elección del ciclo de vida más apropiado para un proyecto es una cuestión fundamental en la estrategia con la que se afronta, ya que incide muy decisivamente en la velocidad con la que se llevará a cabo el proyecto y la satisfacción que generará al cliente. El ciclo de vida claramente dominante hasta hace relativamente poco era el modelo en cascada donde existen las fases de recogida de requisitos, análisis, diseño, codificación, pruebas y mantenimiento del producto. Todas ellas se ejecutan en secuencia, lo que le ha dado a este tipo de ciclo de vida su nombre.

Sin embargo, la evolución de la tecnología y los campos de aplicación y la alta heterogeneidad que presentan los proyectos hoy en día han creado la necesidad de afrontar los proyectos con ciclos de vida personalizados para el perfil de cada proyecto. Por tanto, se discutirán las principales alternativas, exponiendo sus ventajas e inconvenientes.

MODELO EN CASCADA

El modelo en cascada es el ciclo de vida clásico, su principal característica es la naturaleza estrictamente secuencial de la ejecución de sus fases. Al aprobar cada una de ellas se genera la documentación adecuada que permite comenzar con la siguiente, ante defectos que se detectan en la ejecución de una fase determinada posiblemente haya necesidad de volver a la fase inmediatamente anterior y corregir/modificar algunos de sus contenidos, pero es algo que debe evitar en la medida de lo posible. Esta naturaleza se explica con el carácter más homogéneo de las aplicaciones y la plataforma tecnológica mucho más simple de hace unas décadas (las aplicaciones eran prácticamente siempre aplicaciones de gestión sobre host con un nivel de complejidad relativamente simple frente a las actuales).

Este modelo resulta adecuado cuando los requisitos están bien definidos, son estables desde el comienzo del proyecto y se dominan las metodologías y herramientas utilizadas en el proyecto, ya que minimiza el tiempo dedicado a cada una de las tareas.

Ventajas

- Minimiza las tareas de desarrollo repetidas y por tanto el esfuerzo de desarrollo invertido en total
- Minimiza la carga de planificación de los ciclos iterativos de otros ciclos de vida
- Permite afrontar la complejidad de proyectos grandes de una manera muy ordenada y aumenta así las posibilidades de éxito
- Ayuda a trabajar mejor con equipos de desarrollo de relativamente baja cualificación por el alto control de cada actividad y sus resultados

Inconvenientes

- Es muy inflexible, por tanto solamente resulta adecuado cuando hay requerimientos muy bien definidos y muy estables, algo que es difícil de encontrar
- Retroceder en las fases para corregir errores que se han cometido en fases previas o adaptar el proyecto a cambios resulta muy difícil y costoso en esfuerzo

Copyright © 2001 Advanced Quality Solutions

- Aunque la documentación elaborada permite un seguimiento bueno del proyecto para una persona cualificada, los resultados tangibles para el cliente aparecen prácticamente al final del proyecto, algo que muchas veces no aceptan los clientes

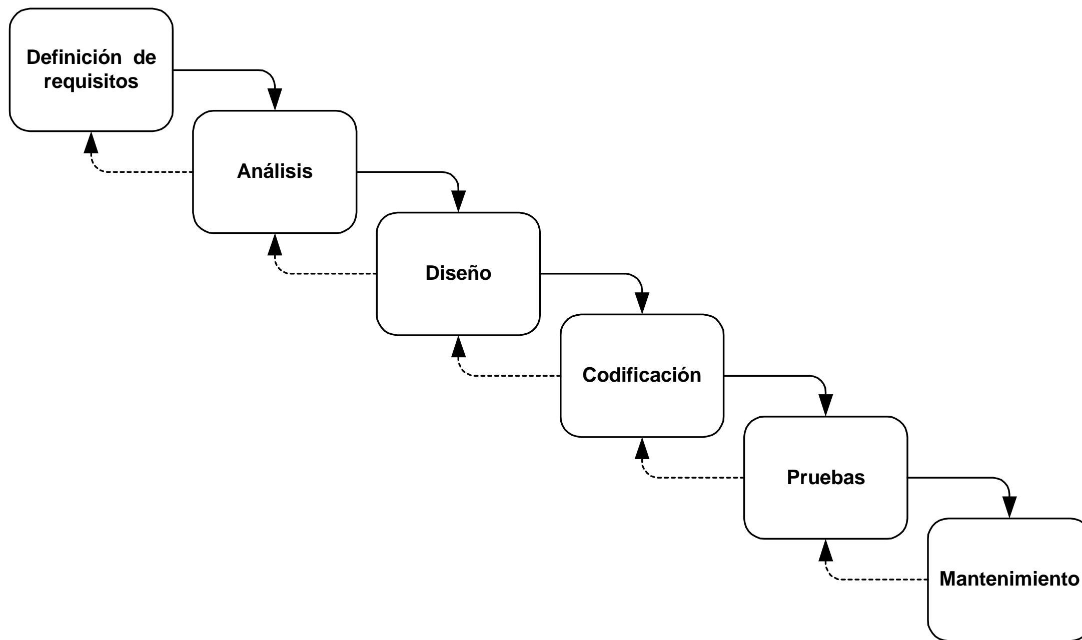

Ilustración 1 – El modelo en cascada. Este modelo es estrictamente secuencial, está permitido retroceder, pero resulta muy costoso y se debería evitar al máximo. Por estas características resulta solamente eficaz en aquellos casos en los cuales los requisitos están muy definidos.

DESARROLLO EN ESPIRAL

El desarrollo en espiral es un ciclo de vida muy orientado a la eliminación progresiva de los riesgos, es un ciclo de vida iterativo en cuyas iteraciones se enfocan uno o más riesgos objetivos que han de superarse hasta que el nivel de riesgo sea suficientemente bajo para continuar con un ciclo menos complejo.

En cada iteración se realizan los siguientes pasos:

- *Planificación:* Determinar objetivos, alternativas y restricciones
- *Análisis de riesgo:* Análisis de riesgos y evaluación de alternativas
- *Ingeniería:* Desarrollo de los entregables o prototipos de la iteración
- *Evaluación del resultado:* Evaluación y validación del resultado

Ventajas

- Puesto que se trata de un modelo orientado a los riesgos del proyecto da un nivel de seguridad muy elevado al proyecto, los riesgos se eliminan al principio que es cuando mejor se puede reaccionar a ellos y en el caso negativo extremo de detectar la inviabilidad del proyecto, minimiza la inversión realizada en él.
- Una mayor inversión en esfuerzo (y con ello tiempo y dinero) se traduce directamente en mayor seguridad del proyecto, ya que permite gestionar con mayor dedicación los riesgos

Copyright © 2001 Advanced Quality Solutions

- El cliente dispone mediante los prototipos de resultados tangibles en cada iteración y participa de una manera muy interactiva en la evolución del proyecto con lo cual se mejoran mucho las posibilidades de satisfacción del resultado

Inconvenientes

- El único inconveniente es la complejidad y carga de gestión de este modelo.

DESARROLLO EVOLUTIVO ORIENTADO A PROTOTIPOS

El desarrollo orientado a prototipos que evolucionan progresivamente no se debe confundir con el modelo en espiral, aunque tienen bastante parecido en cuanto a su carácter iterativo en el modelo en espiral se trata de enfocar los mayores riesgos desarrollando primero las facetas relacionadas con ellos y eliminarlos así cuanto antes, es decir, los riesgos determinan la evolución del proyecto. En el desarrollo evolutivo de prototipos se parte de un concepto inicial de la aplicación y es este concepto el que evoluciona.

Es decir, en este modelo se desarrolla un primer prototipo relativamente completo, frecuentemente destinado a ser ya utilizado por cliente. El cliente aporta realimentación y con ella se desarrolla la siguiente versión, y así sucesivamente hasta que se alcance una versión que le satisface.

Resulta útil cuando el cliente tiene prisa en desarrollar la aplicación, pero no es capaz de definirla con exactitud y el mismo tiene que aprender más de la problemática que debe resolver con la aplicación. También resulta adecuado cuando se prevé que los requerimientos van a tener una tasa de cambio alta durante el desarrollo del proyecto.

Ventajas

- El caso de requerimientos cambiantes e incapacidad de parte del cliente para definirlos con el suficiente detalle se da con frecuencia, abordar el proyecto de esta manera es una solución muy natural ante este problema y evita en gran medida los conflictos con el cliente
- El cliente participa muy activamente en el desarrollo, por tanto, las posibilidades de alcanzar un producto que haga lo "que el quiere" son altas
- Aporta resultados tangibles que permiten al cliente medir el progreso del proyecto
- En muchas ocasiones el cliente gana tiempo en el sentido que ya le resultan útiles los primeros prototipos y amortiza la inversión desde un punto muy temprano mientras que se sigue mejorando el resultado final. Esta faceta frecuentemente hace que el cliente está dispuesto a asumir una inversión global algo mayor a la que estaría dispuesto hacer si tuviera que esperar hasta la entrega final del producto, añade por tanto mucha flexibilidad a la negociación del proyecto

Inconvenientes

- En proyectos de cierta envergadura es prácticamente imposible saber cuando se llegará al producto final, ni cuantos prototipos intermedios serán necesarios hasta entonces
- No es fácil convencer al cliente de la necesidad de tirar determinados prototipos "a la basura", hay una gran tentación de no llegar al final con las iteraciones necesarias

Copyright © 2001 Advanced Quality Solutions

- Desde el punto de vista de los desarrolladores este ciclo de vida puede ser una tentación a desarrollar de forma anárquica, es decir, dejar de lado la modificación de la especificación de requisitos, análisis, etc. que corresponde a cada iteración

EXTREME PROGRAMMING

La metodología de Extreme Programming (XP) es una metodología relativamente nueva y muy polémica porque ataca frontalmente muchos de los principios asentados en la Ingeniería del software y defiende como su nombre bien dice una serie de principios "extremos" y que en su primera lectura parece simplemente completamente irrealistas. Se confunde a menudo con lo que algunas veces se llama "code and fix" en la literatura, es decir, el ponerse a programar directamente, sin un proceso de Ingeniería, muchas veces incluso sin contar ni siquiera con una especificación de requisitos inicial. Téngase en cuenta, además, que no es un ciclo de vida, sino una metodología, pero con un ciclo de vida muy característico.

El principal objetivo de XP consiste en controlar un problema fundamental en el desarrollo de software: la alta volatilidad, especificaciones incompletas cambios y falta de cultura del negocio hace que los desarrolladores tardan en entender el sistema que deben desarrollar. XP ataca este problema mediante ciclos de iteración extremadamente cortos en los cuales se añade muy poca funcionalidad construyendo un sistema completo que posiblemente se utilice incluso en producción, esto permite a los desarrolladores asimilar la problemática de una forma más natural. Por otra parte exige una alta implicación de los usuarios en el desarrollo para crear un autentico clima de trabajo en equipo.

En este sentido el ciclo de vida del Extreme Programming se puede ver como el ciclo de vida orientado a prototipos evolutivos llevado al extremo unido a una serie de principios particulares de esta metodología. Para que el elevado ritmo de integraciones no produzca un sistema degenerado se enfatizan especialmente las pruebas de unidad y prestar un atención alta a la calidad del código, y la necesidad de diseños sencillos. Por otra parte se debe partir de un diseño de base relativamente estable, ya que los fundamentos del diseño no pueden cambiar continuamente.

Pero lo realmente novedoso o podríamos decir, curioso, es el contenido del conjunto relativamente sencillo de reglas en las que el Extreme Programming su método de trabajo. A grandes rasgos:

- Los requerimientos se especifican continuamente y se plasman en las "user stories " que deben ser documentos mucho más ligeros que las especificaciones de requisitos clásicas en los que los usuarios cuentan, en dos o tres fases y un lenguaje no técnico relativamente informal, cada funcionalidad.
- Entregar un número alto de versiones con relativamente pocas funcionalidades nuevas
- Mantener el mayor nivel de rotación entre tareas del proyecto con el fin de que no haya personas que concentren todo el conocimiento de un área y supongan así un alto riesgo. Para que la idea anterior sea factible todo el código de producción debe ser programado por parejas, ya que esto hace que un miembro de la pareja puede cambiar de tarea mientras que el otro da continuidad a la actividad y apoya a la persona que sustituye al primero
- Máxima simplicidad en el diseño
- Utilizar "chapuzas" controladas para reducir riesgos, es decir probar conceptos con pequeños módulos o programas rápidos para medir su viabilidad y tirarlos luego
- No añadir funcionalidad de detalle al principio, mantener la funcionalidad en el nivel más básico posible

Copyright © 2001 Advanced Quality Solutions

- Aplicar técnicas de refactoring siempre que sea aconsejable. Refactoring significa la reconstrucción o reestructuración de código de una manera disciplinada con el fin de eliminar código obsoleto de un proyecto. Es fundamental en una filosofía que parte de una indefinición tan alta como el XP, ya que el diseño detallado y el código tiende a quedarse obsoleto con el tiempo y evolucionarlo significa complicarlo cada vez más. Llegado un momento, resulta más económico empezar desde cero.
- El cliente debe estar siempre disponible.
- Programar antes el código de las pruebas unitarias que el código del producto en sí.
- Propiedad colectiva del código. Esto se refiere a que todo el mundo debe contribuir en la medida de lo posible con ideas en todas las áreas del proyecto. Incluso se concibe que modifiquen código. Esto va unido al hecho de rotar los desarrolladores entre áreas
- Dejar las optimizaciones para el final
- No trabajar en exceso. El razonamiento es que las implicaciones negativas sobre la motivación del equipo descompensan las posibles ganancias de tiempo
- Todo el código debe tener sus pruebas de unidad y no podrá ser puesto en producción hasta pasarlas todas
- Uso intensivo de tests de aceptación

Extreme Programming Project

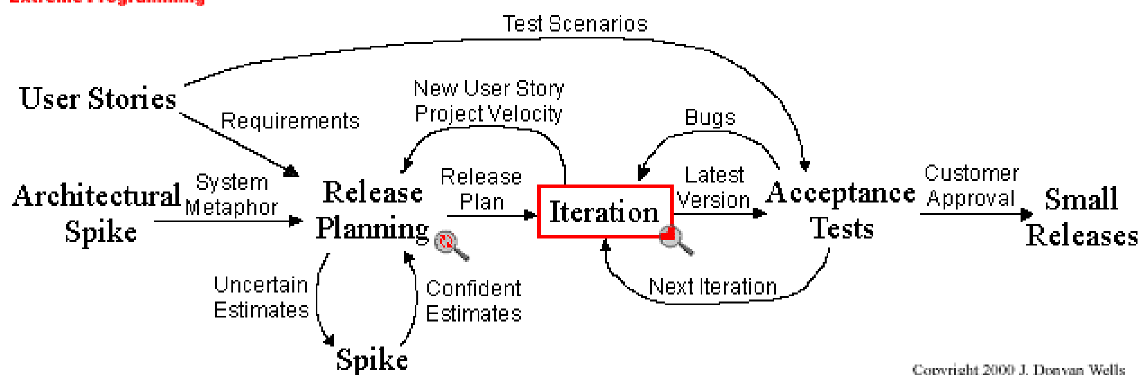

Ilustración 2 - Ciclo de vida de un proyecto al que se aplica la metodología de Extreme Programming.

Copyright © 2001 Advanced Quality Solutions

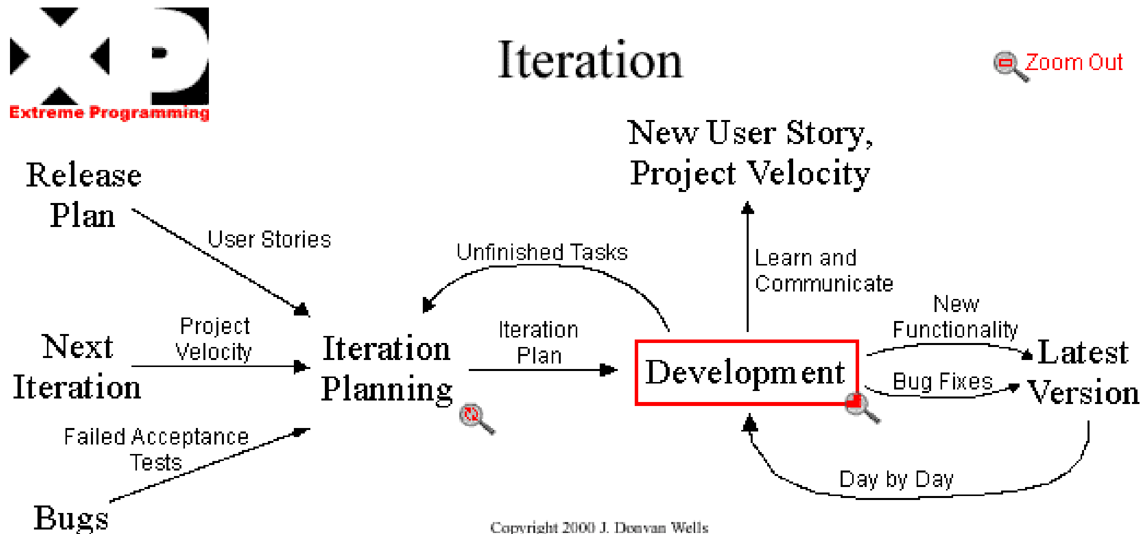

Ilustración 3 – Detalle del trabajo en una iteración en el Extreme Programming. Se aprecia que está orientado a ciclos muy cortos, de días.

La polémica está servida y se centra en dos aspectos principalmente: los costes de desarrollo que genera este modelo de trabajo y la viabilidad de la idea del "código colectivo".

Ventajas

- Parte de una visión muy realista en el sentido que considera la realidad en los desarrollos con los clientes. Así, por ejemplo, tiene en cuenta la dinámica con los clientes en la cual disponer de una definición razonablemente completa de funcionalidad del producto suele ser pura ficción
- Extreme Programming enfatiza especialmente la importancia el diseño simple y la importancia de las pruebas, un punto especialmente descuidado en el desarrollo
- Aporta algunas ideas "refrescantes" que son útiles como principios incluso sin aplicar la metodología en sí. La programación en parejas es un buen ejemplo, o los "stand up meetings"

Inconvenientes

- La metodología no es escalable, solamente puede tener sentido en proyectos relativamente pequeños, según XP proyectos de entre 2 y 12 desarrolladores
- Algunos de los principios parecen demasiado extremos y no gozan de un respaldo con argumentos sólidos por los defensores de esta metodología. Por ejemplo, el principio de la programación colectiva, es decir, que en definitiva todos puedan tocar en todas partes, un punto que genera mucho debate porque el peligro de una evolución anárquica de las funcionalidades de los módulos parece alta y muy difícilmente controlable. Otro ejemplo es el refactoring, la misma metodología pide cierta estabilidad en el diseño base, pero a la vez enfatiza la necesidad de refactoring, dos principios que entran fácilmente en conflicto, pero no aporta criterios claros para saber dónde situar punto de equilibrio entre ambos.
- La metodología no responde con soluciones o con sugerencias a muchos los problemas que plantean sus principios, lo cual da una impresión de encontrarse aún en un estado relativamente hipotético

DESARROLLO ORIENTADO A HITOS

Con frecuencia se da el caso que el factor limitante es el tiempo más que el presupuesto o el detalle de la funcionalidad. En estos casos puede ser muy oportuno aplicar este ciclo de vida que asume ciertas indefinición en la envergadura final de las funcionalidades implementadas, pero fija claramente un punto final en el tiempo. En un desarrollo grande puede ser además muy útil para gestionar el desarrollo de módulos individuales no críticos con el fin de que no retrasen el progreso global del proyecto.

En este ciclo de vida básicamente se trabaja secuencialmente en la base estable del producto que incluye llegar hasta el diseño de la arquitectura, pero a partir de ahí se trabaja en ciclos iterativos sobre el diseño detallado, codificación y pruebas. Los contenidos de estos ciclos se priorizan para optimizar según la relevancia de las funcionalidades implementadas.

Ventajas

- Es una estrategia relativamente óptima ante una situación de fecha límite rígida
- Permite reducir mucho el riesgo en proyectos grandes si se gestionan sus módulos de menor prioridad con esta técnica

Inconvenientes

- Si se consiguen implementar relativamente pocas funcionalidades de las previstas se habrá perdido mucho tiempo en la especificación y diseño de funcionalidades no implementadas al final, por tanto hay que medir bien la ambición del trabajo previo a los ciclos iterativos

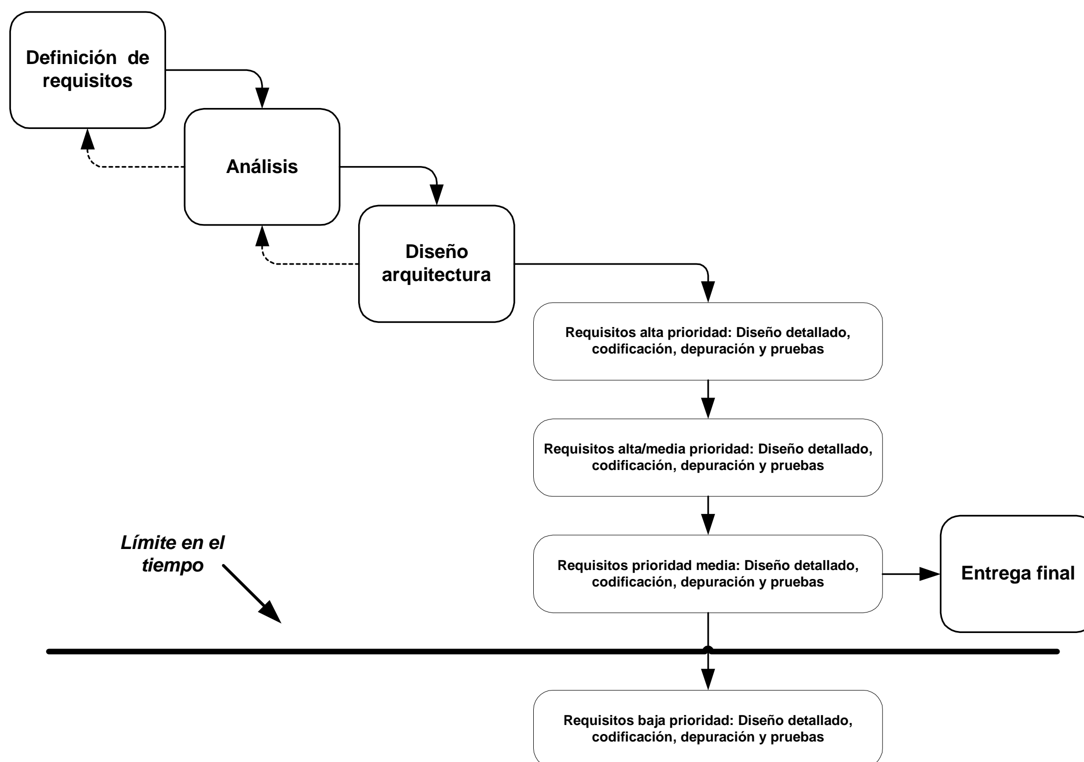

Ilustración 4 – Modelo orientado a hitos. Nótese que se combina en cierta manera el enfoque clásico con el modelo en espiral dividiendo la parte más gruesa del desarrollo en fase priorizadas que permiten optimizar la relevancia del trabajo realizado dentro de unos límites de tiempo, obsérvese especialmente el matiz que la fase de diseño no iterativa se limita al diseño de la arquitectura ya que no varía con las iteraciones.

Copyright © 2001 Advanced Quality Solutions

AQS

MODELOS MIXTOS

Los modelos presentados solo son algunos de los existentes, ante un proyecto concreto cabe la posibilidad de combinar modelos diferentes si el perfil del proyecto lo hace aconsejable, es decir, los modelos presentados no se deben interpretar con un espíritu excesivamente purista, sino tomarlos como estrategias de base ante una serie de características de un proyecto.

Lo importante es que se haga el ejercicio de plantear una estrategia de desarrollo que responda de una manera coherente a la situación previsible del proyecto al que se aplica. Los modelos aquí presentados pueden ser válidos para muchas ocasiones, y si no lo fueran constituirán una "inspiración" para diseñar un modelo a medida más adecuado.

Para ayudar a realizar este ejercicio aportamos una serie de preguntas:

- ¿Cómo de bien entendemos el cliente y yo los requisitos al comienzo del proyecto? ¿Es probable que cambien a lo largo del proyecto?
- ¿Entiendo la arquitectura del sistema? ¿Necesitaré realizar cambios significativos en la arquitectura del sistema durante el desarrollo del proyecto?
- ¿Qué nivel de fiabilidad necesito?
- ¿En qué medida tengo que planificar y diseñar en vista de las futuras versiones?
- ¿Cuál es el nivel de riesgo del proyecto?
- ¿Debo tener en cuenta fechas límite?
- ¿Será necesario disponer de capacidad de maniobra para realizar correcciones en la mitad del proyecto?
- ¿Me va a exigir la dirección resultados intermedios tangibles?

Copyright © 2001 Advanced Quality Solutions

CONCLUSIONES

En proyectos de desarrollo de software habitualmente se dan desvíos sobre la planificación inicial de entre un 25 y 50%, la forma de responder a ello no suele pasar más allá de medidas de mayor presión de calendarios, horas extra y añadir personal a un proyecto. Estudios demuestran que el estrés que los desarrolladores crece continuamente y que, en general, la satisfacción laboral ha bajado comparado con 20 años atrás.

Sin embargo, aunque la situación descrita hoy por hoy aún es una realidad en la mayoría de las empresas, afortunadamente existen ejemplos contrarios que sorprenden por el grado en el que multiplican la productividad de las primeras.

¿Qué es entonces lo que las diferencia? La diferencia que el modo de actuar de las primeras ignoran tanto las características del desarrollo del software en sí como las características de las personas que lo desarrollan. Para las segundas, sin embargo, estos son los puntos de partida para definir su modelo de trabajo y de gestión. Es decir, una empresa de desarrollo que quiere ser efectiva debe tener en cuenta una serie de factores:

- El capital más importante de una empresa de desarrollo son sus recursos humanos, por tanto la gestión debe tener muy en cuenta las características de estos para lograr una máxima productividad
- La tecnología es compleja, por tanto, lo es también el trabajo y las decisiones que se deben tomar en su desarrollo. La correcta gestión supone un reto que no es fácil de alcanzar y en el cual simples decisiones de martillazo serán altamente contraproducentes
- La tecnología tiene un fuerte efecto palanca sobre la productividad, por tanto resulta muy rentable mantener la cualificación de los recursos humanos a un máximo nivel
- Es un trabajo fuertemente creativo, hay que lograr que el entorno de trabajo permita desarrollar la creatividad necesaria
- La actitud ante el trabajo de los desarrolladores es envidiable para muchos otros sectores. Son personas a las que les gusta trabajar y superar retos, por tanto, tienen unos niveles de automotivación muy altos, un punto de partida excelente para lograr una máxima productividad. La empresa que crea un entorno de trabajo que no sea capaz de rentabilizar estas características habrá fracasado en su gestión y modelo de trabajo
- Es realmente sencillo dar a los desarrolladores lo que necesitan para estar motivados, pero exige la sensibilidad suficiente para comprender cómo piensan

En resumen, para enfrentarse con éxito a las exigencias del desarrollo de software hay que conocer muy bien tanto las características del proceso como el perfil psicológico de las personas que realizan esta labor y actuar en consecuencia. Si no es así, aún las mejores herramientas de Ingeniería no servirán de nada.

La clave está en saber explotar la alta motivación que tienen habitualmente de por sí los desarrolladores y gestionar cada proyecto con la estratégica más adecuada a sus características específicas. Tener en cuenta las ideas y principios expuestos en este documento ayudarán sustancialmente a conseguir esta meta, nacen de experiencias contrastadas, tanto en lo que se refiere a éxitos como a fracasos y son por tanto una guía fiable.

Copyright © 2001 Advanced Quality Solutions

AQS

BIBLIOGRAFÍA

A continuación un breve listado de bibliografía interesante relacionada con los contenidos de este documento.

REFERENCIAS EN WEB

- Advanced Quality Solutions. *Sección de downloads en la web, http://www.aqs.es/downloads* - En esta sección se publican periódicamente documentos que reflejan las experiencias, opiniones y resultados de I+D de esta empresa en temas tecnología, gestión y negocio dentro del sector de las tecnologías de la información.
- *http://www.cetus-links.org* - Excelente colección de todos tipo de enlaces relacionado con la Ingeniería del software modera. Tanto a nivel de desarrollo como a nivel de gestión, es el punto de partida perfecto para llegar a todo lo relevante de las tecnologías relacionadas con el software actuales y para estar a la última en la evolución del campo del desarrollo del software.
- American Management Association. http://www.amanet.org/index.htm - En este sitio se encontrarán artículos interesantes sobre todo tipo de temas relacionados con la gestión de proyectos, referencias bibliográficas sobre temas específicos, calendarios de eventos, etc.
- Eric Raymond. *The Cathedral an the Bazaar*. http://www.openresources.com/documents/cathedral-bazaar/ - Un clásico de obligada lectura para cualquiera que tenga interés en la gestión de proyectos, para dos estilos opuestos de desarrollo, el modelo que el llama el "modelo de catedral", propio de la mayoría de los desarrollos con el "modelo de bazar" propio del mundo Linux. Como figura importante en el mundo del Open Source es evidente hacia qué modelo se inclina, y lo hace con un argumento de partida contundente: si ha sido posible desarrollar un sistema operativo de la categoría de Linux con participantes en todos los lugares del mundo el modelo del bazar debe tener algo de sentido.
- Extreme Programming. http://www.extremeprogramming.org/ - Sitio de referencia para conocer más sobre este curioso modelo que ha aparecido recientemente, incluimos esta referencia sobre todo por la actualidad de la polémica sobre esta metodología.
- Refactoring. http://www.refactoring.com/ - Sitio web sobre esta técnica reestructuración disciplinada de código.

LIBROS & ARTÍCULOS IMPRESOS

Libros de referencia

- Ian Sommerville. *Software Engineering, 6th Edition.* Addison Wesley 2000.

Copyright © 2001 Advanced Quality Solutions

- Kent Beck. *Embrace Change.* Addsion Wesley 1999 – Este libro es una introducción a esta metodología polémica que ha surgido con fuerza en los últimos dos años, de hecho aporta una manera de pensar alternativa con algunos aspectos muy interesantes. El libro está dividido en tres apartados que explican en qué consiste esta metodología, expone cómo aplicarla y el último se centra en cómo introducir esta metodología en una empresa.
- Roger S. Pressman. *Software Engineering – A Practioneers Approach, 5th Edition.* McGraw Hill 2000.
- Steve MCConnell. *Rapid Development*. Microsoft Press 1996 – Excelente libro que se centra en como conseguir desarrollos de aplicaciones lo más eficientes y por tanto rápidos posibles. La fórmula propuesta y plenamente acertada es buena gestión a todos los niveles: planificación, riesgos, personal, etc. Expone cómo llevar esta gestión a la práctica aportando la experiencia de más de 20 años de su autor en este campo. No en vano es la fuente de inspiración principal de este documento.

Artículos y fuentes varias

- Barry W. Boehm. *Software Engineering Economics.* Englewood Cliffs, N.J.: Prentice Hall 1981
- Barry W. Boehm. *Software Risk Management.* Washington, D.C.: IEEE Society Press 1989.
- Bill Curtis et al. *Software Psychology: The Need for an Interdisciplinary Program.* Proceedings of the IEEE 1986, vol. 74, no. 8 (August): 1092-1106
- Bill Curtis. *Substantiating Programmer Variability.* Proceedings of the IEEE 1981, vol. 69, no. 7 (July): 846
- Capers Jones. *Assesment and Control of Software Risks.* Englewood Cliffs, N.J. Yourdon Press 1994.
- Carl E. Larson, Frank M.J. LaFasto. *Teamwork: What Must Go Right; What Can Go Wrong.* Newbury Park, Calif.: Sage. 1989.
- David N. Card. *A Software Technology Evaluation Program.* Information and Software Technology 1987, vol. 29, no. 6 (July/August): 291-300
- Gerald M. Weinberg. *The Psychology of Computer Programming.* New York: Van Nostrand Reinhold.
- H. Sackman, W.J. Erikson, E.E. Grant. *Exploratory Experimental Studies Comparing Online and Offline Programming Performance.* Communications of the ACM 1968, vol. 28, no. 3 (Fall): 403-424
- Harlan D. Mills. *Software Productivity.* Boston, Mass. 1983: Little Brown. 71-81.
- J. Richard Hackman, Greg R. Oldham. *Work Redesign.* Reading, Mass. 1980: Addison Wesley.
- J. Valet, F.E. McGarry. *A Summary of Software Measurement Experiences in the Software Engineering Laboratory.* Journal of Systems and Software 1989, 9 (2): 137-148
- James Bach. *Enough About Process: What We Need Are Heroes.* IEEE Software, Marzo 1996-98.
- Larry L. Constantine. *Constantine on Peopleware.* Englewood Cliffs, N.J.: Yourdon Press 1995
- Rob Thomsett. *Project Pathology: A Study of Project Failures.* American Programmer. Julio 1995, 8-16.

- Tom DeMarco, Timothy Lister. *Peopleware: Productive Projects and Teams.* New York: Dorset House. 1987
- Tom DeMarco, Timothy Lister. *Programmer Performance and the Effects of the Workplace. Proceedings of the 8th International Conference on Software Engineering.* Washington, D.C.: IEEE Computer Society Press 1985, 268-272

Copyright © 2001 Advanced Quality Solutions