**Creando una Cultura de Ingeniería de Software**

Rara vez en la historia ha evolucionado un campo de trabajo tan rápido como el desarrollo de software en este momento.

La lucha por estar al tanto de las nuevas tecnologías, lidiar con los retrasos acumulados en el desarrollo y lidiar con los problemas de las personas se ha convertido en una carrera sin fin, ya que los grupos de software trabajan tan duro como pueden para mantenerse en su lugar.

El Software Engineering Institute (SEI) y hordas de gurús nos exhortan a mejorar nuestro proceso de desarrollo, pero ¿cómo podemos permitirnos el tiempo? No todos los miembros de una organización sienten la necesidad de cambiar. Es muy fácil descartar los esfuerzos de mejora de procesos como la última crítica de la administración. Ahí radica la semilla del conflicto, ya que algunos miembros de un equipo adoptan nuevas formas de trabajar, mientras que otros murmuran "sobre mi cadáver".

La cultura de una organización es un factor de éxito crítico en sus esfuerzos de mejora de procesos.

"Cultura" incluye un conjunto de valores y principios compartidos que guían los comportamientos, actividades, prioridades y decisiones de un grupo de personas que trabajan en la misma área. Cuando los compañeros de trabajo se alinean según las creencias comunes, es más fácil inducir cambios que aumentarán la efectividad del grupo y su probabilidad de supervivencia. Una cultura compartida es una diferencia entre un "equipo" y un "grupo de bozos en un autobús" (parafraseando a Larry Constantine).

Este artículo describe parte de la filosofía compartida, la cultura, que evolucionó durante varios años en un pequeño grupo de software en una corporación muy grande. Nuestro grupo desarrolla aplicaciones personalizadas para apoyar la investigación fotográfica en Eastman Kodak Company. Describiré cómo cada creencia o valor nos influye para actuar de cierta manera a medida que tratamos continuamente de mejorar la forma en que construimos los sistemas de software. Creemos que esta cultura ha mejorado nuestra efectividad como ingenieros de software, la relación y la reputación que tenemos con nuestros clientes, nuestro nivel de trabajo en equipo y el placer que obtenemos al venir a trabajar todos los días. Nuestro enfoque general para implementar la ingeniería de software en este pequeño grupo se describió anteriormente en Computer Language.

Ahora estamos en nuestro cuarto año de mejora continua de procesos. Durante ese tiempo, el grupo se ha más que duplicado en tamaño, algunos roles han cambiado y las pocas personas que no respaldaron la cultura común han avanzado. Compartir nuestra filosofía con los candidatos durante las entrevistas facilitó la selección de aquellos que se asimilarían fácilmente al equipo y apoyarían los valores que hemos acordado. Y tiene que ser más fácil ingresar en una organización que ha pensado en lo que cree que lanzarse ciegamente al fuego del trabajo y ver cómo funciona.

**La participación del cliente es el factor más crítico en la calidad del software.**

Cualquier empresa exitosa requiere que ambos hagan lo correcto y hagan lo correcto. Esperamos que los desarrolladores profesionales de software sepan cómo hacer lo correcto.

Hacer lo correcto, sin embargo, requiere una comprensión inequívoca de lo que espera su cliente. Con este fin, nos esforzamos por maximizar la participación de nuestros clientes en nuestras actividades de desarrollo.

Involucrar a los clientes en el desarrollo es sin duda más sencillo en una situación como la nuestra, donde nuestros "clientes" son compañeros de trabajo, generalmente en el mismo edificio en que vivimos. El problema de la participación del cliente es más desafiante cuando escribe para el mercado comercial; la participación de tipos de mercadeo puede confundir aún más el problema. Pero para el desarrollo interno o incluso contratando trabajo, no hay excusa para no involucrar a los usuarios finales desde el primer día.

Hemos formalizado nuestras expectativas para la participación del cliente al exigir que cada proyecto tenga uno o más "campeones del proyecto" de la comunidad de usuarios. El campeón del proyecto actúa como nuestra principal interfaz para los grupos de clientes que son usuarios potenciales del sistema que se está creando. Simplemente nos negamos a trabajar en un proyecto si la administración de clientes no está dispuesta a comprometer a los campeones de proyectos adecuados para el equipo de desarrollo. Nuestras directrices de desarrollo de software incluyen las expectativas escritas del campeón del proyecto, pero los detalles son negociables.

Además del valor obvio de tener la voz del cliente directamente disponible para el equipo del proyecto, hemos descubierto un beneficio adicional del modelo de campeón del proyecto. Los campeones obtienen una idea de nuestro enfoque de desarrollo de software estructurado y, por lo general, se convierten en firmes defensores de nuestro proceso con sus colegas y gerentes. Los campeones entienden que el proceso que seguimos está dirigido a construir el producto correcto de la manera más eficiente posible. Estos miembros influyentes de la comunidad de usuarios ayudan a suavizar la naturaleza a veces antagónica del desarrollo de sistemas.

**Nuestro problema más difícil es compartir la visión del producto final con el cliente.**

La falla clásica del desarrollo de software es que el producto entregado solo coincide vagamente con las expectativas del cliente (suponiendo que realmente se entregue algo). Nuestra primera actividad de mejora de procesos se centró en mejorar nuestro proceso de especificación de requisitos, en un esfuerzo por reducir esta brecha de expectativas. Todos los proyectos, incluso los pequeños que desarrollan componentes de software reutilizables internamente, se someten a un proceso formal de especificación de requisitos escritos. También hacemos un uso extensivo de prototipos de interfaz de usuario verticales y horizontales para ayudar a refinar los requisitos del usuario y explorar alternativas de diseño.

"Formal" no necesariamente significa "engorroso, voluminoso, difícil de manejar, incomprensible, inclusivo, inmutable e interminable". Nuestro objetivo es simplemente preparar un documento explícito, consistente, inequívoco y estructurado que identifique claramente las características que deben estar contenidas en el sistema entregado para satisfacer las necesidades del usuario. Por supuesto, los requisitos con frecuencia deben modificarse a medida que avanza el desarrollo. Esto está bien, siempre que todos los involucrados entiendan el impacto y estén de acuerdo con los cambios

Recientemente adoptamos el estándar IEEE para especificaciones de requisitos de software (SRS), que se adapta muy bien a nuestras necesidades. Un IEEE SRS contiene secciones para capturar muchas de las piezas de información que finalmente se necesitan para construir el sistema: restricciones, interfaces externas y atributos de calidad, además de los requisitos funcionales reales.

Los campeones del proyecto son participantes clave en la creación del SRS. Típicamente, uno de los miembros de nuestro equipo hace la mayor parte de la redacción actual, pero es realmente un esfuerzo colaborativo. Los campeones son responsables de resolver ambigüedades y conflictos con los usuarios finales que representan, de modo que el SRS contiene un único conjunto de requisitos unificados. Nos negamos a quedar atrapados en la trampa en la que se deja al programador, resolver los requisitos conflictivos y negociar acuerdos con los diferentes clientes involucrados. Raramente estamos calificados para tomar decisiones orientadas a los negocios, por lo que se los dejamos a los campeones. Los campeones ayudan a determinar qué funcionalidad agrega más valor al sistema mientras controla el cromo. El modelo de campeón del proyecto es una piedra angular de nuestra cultura de ingeniería

Recientemente comenzamos a incluir requisitos de atributos de calidad más específicos en el SRS, tales como mantenibilidad, extensibilidad, portabilidad, confiabilidad, reutilización y rendimiento. El proyecto los campeones están jugando un papel cada vez más importante al aceptar estos requisitos menos tangibles. Cuantificando estos atributos en el SRS obligan al cliente a pensar en ellos antes de tiempo, en lugar de que esperar hasta que se entregue el sistema y estar decepcionado porque una necesidad importante era nunca discutida explícitamente.

La calidad del documento de requisitos aún debe ser verificada. Cada SRS se somete a un

una inspección exhaustiva por parte de un grupo compuesto por los desarrolladores involucrados, los campeones del proyecto, un representante de garantía de calidad de software de nuestro equipo, y al menos una persona externa que tiene ninguna participación específica en el proyecto. El extraño puede ser otro de los miembros de nuestro equipo, alguien de un grupo de software diferente, o cualquier otra persona que pueda evaluar críticamente el SRS desde una perspectiva imparcial. Los defectos identificados durante la inspección se clasifican y registran según corresponda acción.

Hay quienes sienten que esta gran estructura en torno a la especificación de requisitos es innecesaria o incluso contraproducente. Por el contrario, creemos que es la base de la calidad en los sistemas exitosos que entregamos.

Nuestro grupo han estado midiendo las distribuciones de esfuerzo de trabajo en diferentes fases de desarrollo y mantenimiento de todos los proyectos desde 1990. Nuestros datos indican que gastamos un promedio del 21% de nuestro esfuerzo de desarrollo total en las especificaciones de los requisitos.

Yo creo que esta es una inversión razonable para "hacer lo correcto". Para esos clientes, gerentes, o desarrolladores que afirman que no podemos pasar tanto tiempo en las especificaciones, cito el antiguo signo del laboratorio de química: "Si no tienes tiempo para hacerlo bien, ¿cuándo tendrás tiempo para hacerlo?"

**La calidad es la principal prioridad; la productividad a largo plazo es una consecuencia natural de alta calidad.**

Este principio parece evidente, ya que la calidad está de moda en todas las empresas en estos días. Pero también muchos esfuerzos de calidad se sacrifican en el altar de la productividad, como gerentes y trabajadores por igual afirman que podrían producir más si no tuvieran que hacer todo esto relacionado con la calidad.

En el software, la relación entre la calidad y la productividad se reduce al trabajo frente a la repetición, también conocida como "mantenimiento". La filosofía de nuestro grupo es que el tiempo que invertimos en la creación de un producto de calidad en primer lugar se amortiza ampliamente a lo largo de la vida útil del producto a través de costos de mantenimiento más bajos.

El beneficio menos tangible es la buena voluntad del cliente. Creemos que la mayoría de los clientes están dispuestos a esperar un poco más para obtener un mejor producto que les ayude a hacer su trabajo correctamente sin perder el tiempo debido a fallas y mal uso. Nuestras experiencias de proyectos recientes respaldan esta afirmación.

Nuestras métricas de esfuerzo de trabajo refuerzan primero el concepto de calidad. Clasificamos el mantenimiento en cuatro categorías: correctiva (corregir errores), perfeccionar (agregar mejoras), adaptativa (modificaciones en respuesta a un entorno cambiante) y soporte para el usuario. Desde que comenzamos nuestro programa de mejora de procesos, la fracción de nuestro tiempo de trabajo total que se dedica al mantenimiento correctivo ha disminuido a un estado constante de alrededor del 2%, una fuente de considerable orgullo para nosotros. Tener estos datos disponibles nos ayuda a identificar dónde va nuestro esfuerzo de mantenimiento para poder centrar mejor las actividades de mejora para el mayor apalancamiento. Cada hora que no gastamos en arreglar un programa existente es una hora en la que podemos dedicarnos a escribir algo nuevo para ayudar a nuestros clientes a hacer mejor su trabajo.

Por supuesto, este no es un pozo de oportunidad infinitamente profundo. Una vez que el mantenimiento se reduce a un nivel aceptablemente bajo, debemos buscar mejoras de productividad adicionales en otros lugares.

Otra forma de pensar sobre la calidad es considerar que cada paso en el proceso de desarrollo de software es el "cliente" del paso anterior. Por ejemplo, los productos creados durante la especificación de requisitos se convierten en las materias primas para el diseño. Esto es cierto para cualquier ciclo de vida de desarrollo de software, ya que todos (cascada, espiral, evolutivo, orientado a objetos, etc.) involucran las tareas de especificación, diseño, implementación y prueba. La Figura 1 muestra una forma de describir las relaciones cliente / proveedor del proceso de desarrollo de software. La calidad del producto que se puede alcanzar en cualquier paso está limitada por la calidad de las materias primas que se le suministran. Esta perspectiva lo alienta a adoptar prácticas que aseguren la calidad de los entregables producidos en cada paso del ciclo.

**La mejora continua de su proceso de desarrollo de software es posible y esencial.**

El Software Engineering Institute define una escala de madurez de procesos de software, que va del nivel 1 (inicial) al nivel 5 (optimización). Prácticamente ninguna organización está funcionando en los niveles 4 o 5, aunque algunos proyectos individuales se han medido en esos niveles. La premisa detrás de la escala de SEI es que una organización de software que mejora continuamente la forma en que hace su trabajo obtendrá mejores resultados que una organización en la que cada miembro hace las cosas a su manera.

Mientras que pocos de nosotros es probable que alcancemos el nivel estratosférico 5 en el corto plazo, cada grupo de software tiene oportunidades de mejora. Deberíamos buscar constantemente para adquirir y compartir las mejores prácticas en las muchas subdisciplinas que constituyen el ciclo completo de desarrollo de software.

Una cultura en la que se fomenta el intercambio de ideas y prácticas, en la cual la educación continua es apoyada y recompensada, escalará la escala de madurez del proceso más rápido.

Creemos que la mejora del proceso debe ser evolutiva, no revolucionaria, pero su cultura puede ser más audaz. Puede comenzar con una sesión grupal de lluvia de ideas para identificar las oportunidades de mejora más urgentes. Luego, diseñe un plan simple para implementar estas mejoras, con fechas objetivo explícitas, entregables claros y responsabilidades específicas aceptadas por los diferentes miembros del equipo. La participación de todos los miembros del equipo crea un mayor sentido de propiedad del proceso y los resultados. Imponer cambios por mandato de la gerencia debería ser el último recurso. Su proceso debe ser reexaminado periódicamente para juzgar su efectividad (quizás con la ayuda de datos de métricas) y descubrir nuevas oportunidades de mejora o necesidades críticas.

También nos pareció valioso establecer objetivos anuales de mejora de la calidad del equipo. Cada año, el equipo identifica colectivamente de seis a diez objetivos específicos en cuatro o cinco áreas clave de resultados. Se acuerdan los objetivos cuantitativos y hacemos un seguimiento del progreso durante todo el año. Hemos logrado que estos objetivos sean los impulsores para cambiar la forma en que hacemos nuestro trabajo, en lugar de simplemente ser objetivos que esperamos alcanzar mágicamente a finales de año. La selección y el logro de los objetivos de mejora mensurables y significativos también envía un mensaje a nuestra gerencia de que nos tomamos en serio la mejora de la forma en que hacemos nuestro trabajo.

La cultura es un factor de éxito vital en la introducción de programas de métricas de software, una característica de las organizaciones más maduras. Los miembros del equipo deben confiar en que los datos no se utilizarán en su contra y que se utilizarán para obtener algún beneficio tangible. Nuestro éxito en la recolección y el uso de las métricas de esfuerzo de trabajo se basa en la idea de que los datos pertenecen a los miembros del equipo, no al líder del equipo. El líder comparte los resúmenes de datos para que todo el equipo pueda llegar a conclusiones y acordar los cambios del proceso en función de los datos, así como compartir los éxitos indicados por las métricas recopiladas. Me temo que los programas de métricas de software pueden fallar en las organizaciones que carecen de esta confianza mutua y compromiso con las mejoras de calidad.

**La educación continua es responsabilidad de cada miembro del equipo**

Mientras llenaba algunos puestos nuevos en nuestro equipo recientemente, le pregunté a cada candidato cómo se mantenía con la literatura de software. Las respuestas fueron desalentadoras. Algunos no entendieron la pregunta.

Pocos habían leído los libros de software últimamente. Muy pocos estaban en programas de grado avanzados. La mayoría de las revistas leídas eran publicaciones de libre comercio, no publicaciones periódicas técnicas. Casi ninguno pertenecía a organizaciones informáticas profesionales como la IEEE Computer Society o la Association for Computing Machinery.

Nuestro grupo se suscribe a más de 15 publicaciones de informática técnica, que cubren una amplia gama de temas: bases de datos, programación C, tecnología de PC, UNIX, mainframe, ingeniería de software, calidad de software, etc. Compramos (y leemos) muchos libros. Asistimos (y presentamos) en conferencias. Varios miembros del equipo adquirieron títulos de pregrado o avanzados en informática o ingeniería de software a través de estudios nocturnos. Cada miembro del equipo puede obtener capacitación formal en habilidades directamente relevantes para las actividades laborales a expensas de la compañía.

Algunas de estas actividades de mejora son costosas, pero muchas no lo son. En un área técnica rápidamente cambiante como el desarrollo de software, cada profesional debe pasar tiempo estudiando la literatura publicada y buscando formas de mejorar lo que hace. Nuestra cultura también fomenta el intercambio de ideas, la lluvia de ideas y la consulta con colegas para obtener asesoramiento. Todos se sienten cómodos acercándose a los demás en busca de información técnica y crítica informal de los productos del trabajo. Un entorno competitivo en el que las personas son reacias a compartir sus conocimientos no es propicio para mejorar el rendimiento del equipo.

El administrador de software no está exento del desafío de aprendizaje continuo. Mucho se ha publicado sobre la administración de proyectos de software, pero las escrituras sobre administración de personas de software son más difíciles de encontrar. Un excelente lugar para comenzar es con DeMarco y Lister's Peopleware. Este pequeño libro es una lectura fácil con una gran cantidad de información sobre la importancia de los problemas humanos en la calidad del software, la productividad y la satisfacción laboral. También recomiendo The Decline and Fall of the American Programmer, de Ed Yourdon. No pierda el interés o el pánico después de leer el primer capítulo muy publicitado sobre el tema del título. El resto del libro es un excelente tratado sobre prácticas contemporáneas de desarrollo de software de alta calidad que todos deberíamos implementar.

**Preferimos que un par, en lugar de un cliente, encuentre un defecto.**

Todos hemos recibido llamadas de clientes iracundos, quejándonos de que los errores en nuestros programas perdían su tiempo, daban resultados incorrectos o los agravaban. La filosofía de nuestro grupo es que preferiríamos sufrir la ligera vergüenza de que otro miembro del equipo encuentre un defecto en un diseño o programa que tener que enfrentar a un usuario con homicidio en su mente.

Este valor se traduce en la voluntad de que nuestros productos de desarrollo sean inspeccionados formalmente por sus pares en un esfuerzo por identificar fallas (defectos) antes de que causen fallas. Todos los tipos de entregas se someten a inspecciones: especificaciones de requisitos, modelos de diseño, código, planes de prueba, documentación del sistema, manuales de usuario. Aunque no se inspeccionan todos los productos entregados, la mayoría de nosotros ahora se siente un poco nerviosa si no hemos tenido a alguien que revise nuestro trabajo antes de continuar.

Las inspecciones de especificación tienen el mayor apalancamiento, ya que un error corregido en esa etapa no necesita ser corregido, a un costo mucho más alto, cuando finalmente se encuentra más adelante.

Existe considerable evidencia en la literatura de software de que las inspecciones son la actividad de calidad más efectiva que puede realizar. Si no sabe cómo comenzar, intente con el sistema de amigos. Haga una pareja con uno de sus socios y ofrezca revisar sus productos si va a revisar los suyos. Se han publicado varios libros que describen técnicas para revisiones, inspecciones y recorridos.

La cultura de una organización es muy importante para determinar si las inspecciones serán exitosas. De nuevo, necesitas una atmósfera de confianza y respeto mutuo.

Los participantes deben tener cuidado de criticar el producto y no el productor.

Por el contrario, el productor no debe ver las sugerencias de mejora como una crítica de sus habilidades. Debería haber una voluntad de aprender el uno del otro. He aprendido algo de cada inspección en la que he participado, ya sea como revisor o un desarrollador.

No es fácil poner tu trabajo sobre la mesa para que tus compañeros lo vean y mastiquen. Pero si puede aprender a controlar sus egos en la puerta, cada inspección que realice le ayudará a entregar un mejor producto a sus clientes y ayudará a los participantes a mejorar sus habilidades de software.

Mi filosofía es "inspeccionar temprano y a menudo". Los programadores a menudo son reacios a permitir que un compañero vea un producto incompleto, y prefieren terminarlo antes de exponerlo a una crítica constructiva.

Creo que esto es un error. Recuerde, el apalancamiento proviene de encontrar problemas lo más temprano posible.

Hay dos razones para alentar a los desarrolladores a enviar materiales para la revisión por pares al inicio del juego. En primer lugar, si una inspección de una versión preliminar o parcial de un documento o programa encuentra alguna oportunidad de mejora sistemática (como en el caso de la programación o el estilo de escritura), los cambios necesarios se pueden implementar desde el principio. Sin embargo, si se encuentran los mismos problemas solo después de escribir 5.000 líneas de código, la cantidad de reprocesamiento puede ser tan abrumadoramente grande que los cambios nunca se realicen.

En segundo lugar, mucho más de nuestro ego está ligado a un producto terminado que a una versión preliminar. Por lo tanto, somos más resistentes a hacer cambios si se supone que el elemento que se está inspeccionando es el producto final. Intente forzarse a sí mismo para permitir que otros examinen productos de trabajo que aún se encuentran en una etapa donde las sugerencias son bienvenidas. Simplemente informe a los revisores que el producto que se inspecciona no está en su forma definitiva, para que sepan cómo evaluarlo.

**Los procedimientos escritos de desarrollo de software pueden ayudar a mejorar la calidad.**

Una clara indicación de la madurez del proceso de una organización de software es si utiliza un proceso reproducible y documentado para cada nuevo proyecto. Una de nuestras actividades más tempranas de mejora de procesos fue escribir un conjunto de procedimientos de desarrollo de software concisos.

Estas son pautas, no leyes: si hay una MUY BUENA razón para hacer algo diferente a lo que indican las pautas, está bien. Pero debería ser una MUY BUENA razón, no un capricho personal.

Nuestros procedimientos totalizan menos de 50 páginas, pero abordan la mayoría de los aspectos importantes de nuestro trabajo. Lo más importante es que las personas realmente los usan. Puede alentar esto de maneras sutiles: cuando alguien le pregunta cómo debe hacer algo, pregúntele qué dicen sus procedimientos al respecto.

No caiga en la trampa de "no inventar aquí" cuando se trata de procedimientos de escritura (o cualquier otra cosa, para el caso).

Si está en una empresa de buen tamaño, probablemente ya haya muchos conjuntos de procedimientos departamentales de los que pueda tomar prestado o simplemente adoptar.

El IEEE publica muchos estándares integrales de desarrollo de software. Recientemente hemos comenzado a seguir los estándares IEEE para varios aspectos del desarrollo, incluidas especificaciones de requisitos de software, documentación de prueba, planes de garantía de calidad y seguimiento de anomalías (defectos, mejoras e informes de incidentes).

La existencia de directrices de desarrollo de software ayuda a construir la cultura compartida de prácticas y expectativas que mejora el rendimiento de un equipo. Esto es particularmente cierto si los miembros del equipo están involucrados en la escritura, la crítica y la selección de las pautas.

Planifique revisar y actualizar sus pautas periódicamente como parte de sus actividades de mejora continua.

**La clave del éxito del software es iterar tantas veces como sea posible en todos los pasos de desarrollo excepto en la codificación: haz esto una vez.**

Existe amplia evidencia en la literatura de software que es mucho más costoso corregir un defecto en un producto entregado que si el defecto se encuentra en una etapa de desarrollo temprano.

Basándome en esa información y en mi propia experiencia, me he convencido de que debemos intentar escribir el código solo una vez, haciendo lo que sea necesario antes de la implementación para asegurarnos de que estamos escribiendo correctamente el código correcto.

Con este fin, invertimos mucho trabajo en perfeccionar nuestras especificaciones y diseños, sin caer en la trampa de la "parálisis analítica". Me imagino que 10 horas adicionales dedicadas a la mejora de las especificaciones ahora pueden ahorrarse fácilmente 100 horas más tarde si no me permite entregar el código incorrecto.

Esta es también la razón por la que creo en el valor de construir modelos de nuestros sistemas antes de escribir el código.

Hacemos un uso extensivo de las herramientas CASE para el modelado de aplicaciones para el usuario, particularmente diagramas de flujo de datos, relación entre entidades y diagramas de transición de estado para modelos de interfaz de usuario (mapas de diálogo).

El poder real de las herramientas CASE es la facilidad con la que puedes modificar un diagrama cuando ves una oportunidad para mejorarlo. No podemos hacerlo bien la primera vez.

En cambio, tratamos de acercarlo la primera vez e iterar, con la opinión de los demás, hasta que el modelo sea tan bueno como podamos. ¿Cómo sabes cuándo parar? Cuando no sigues encontrando otro cambio para hacer debajo de cada roca, y cuando las particiones y las interfaces de proceso u objeto "se sienten bien" (esto requiere algo de experiencia).

Pienso en el uso de métodos estructurados de desarrollo de software como una forma de evitar sorpresas.

Nadie quiere el tipo de sorpresa que comienza cuando un usuario dice: "Pero pensé que se suponía que ...". La iteración en los requisitos (enunciado del problema) y en el diseño (solución propuesta para el problema) ayuda enormemente a controla el factor sorpresa.

**Nunca permita que su jefe o su cliente lo convenzan de que haga un mal trabajo.**

De vez en cuando, puede que se le pida que reduzca la calidad del trabajo que hace. A veces, el solicitante es un supervisor que no aprecia el valor de las actividades de calidad que usted practica o que siente las presiones del presupuesto o del mercado que usted no tiene. A veces es un cliente el que quiere que se concentre en sus necesidades específicas cuando ha identificado la oportunidad de generalizar más allá de sus requisitos para proporcionar un valor mejorado a una comunidad de usuarios más amplia.

Me gustaría pensar que estamos más allá del escenario de "no te molestes con esas especificaciones, comienza a escribir código", pero me temo que esta llamada del software de dinosaurio todavía se hace eco en toda la tierra.

No es fácil resistir estas presiones de las personas que pagan la factura o su salario.

A veces no tienes elección. Pero en una cultura de software de calidad, se seguirán prácticas estándar tanto en tiempos de crisis como en tiempos normales. Hemos tratado de adoptar estándares personales que nos motiven a mantener nuestras armas de procesamiento de software en el fragor de la batalla. Considera educar a tu jefe para que aprecie mejor el valor de tu enfoque disciplinado. Por supuesto, puede que necesite atemperar este idealismo con la realidad de mantener su trabajo, pero preferimos comenzar desde la posición de calidad *uber alles.*

**La gente necesita sentir que el trabajo que hacen se nota.**

Cuando era el supervisor de nuestro equipo de software, inicié un programa de reconocimiento simple. Cuando alguien alcanza un hito menor en su proyecto o hace una contribución como ayudar a un compañero de equipo con un problema, le di un pequeño paquete de M & Ms, con una etiqueta adjunta en la que expresaba mis felicitaciones o agradecimiento, según correspondía.

Mayores logros generaron bolsas más grandes de M & Ms o premios de reconocimiento más sustanciales.

Como esperaba, los dulces desaparecieron de inmediato, pero me sorprendió gratamente ver que la gente mantenía visibles las etiquetas en sus escritorios. La parte importante no era la bolsa de dulces de 50 centavos, sino las palabras que indicaban que noté y valoré el progreso que estaban logrando los miembros de mi equipo. También di este tipo de reconocimiento de micro reconocimiento a personas ajenas al equipo que nos ayudaron de alguna manera. Trajo sonrisas a sus rostros y buena voluntad para nuestras relaciones.

M & Ms ya eran algo así como una broma en nuestro grupo; alguna otra técnica de reconocimiento social podría funcionar mejor para usted.

Curiosamente, los propios miembros del grupo indicaron que prefirieron que las presentaciones de M & M se hicieran públicamente en nuestras reuniones semanales de personal, lo que indica el deseo de reconocimiento de iguales incluso de pequeños logros.

También pasamos unos momentos en reuniones semanales de personal para darles a todos los miembros del equipo la oportunidad de transmitirles a los demás "refuerzo positivo".

Independientemente de cómo lo elija, parece que hay algún tipo de elogio y elogio público

ayudar a construir el espíritu de esfuerzo por la excelencia que todos queremos en nuestros equipos.

**Haz lo que tiene sentido: ningún dogma.**

Libros y artículos sobre metodologías de desarrollo de software abundan.

Un error común cometido por las organizaciones que intentan mejoras en el proceso es adoptar algún bloqueo de metodología publicado, stock y barril.

Esta es una receta para el tiempo perdido, el dinero desperdiciado, el resentimiento y la sensación de que el cambio de proceso no es posible en su equipo.

Toda metodología tiene buenas ideas e ideas tontas, en proporciones variables. Mi consejo es seleccionar cuáles parecen ser las mejores ideas que puedes encontrar e intentar implementarlas de forma no amenazante, pero de manera significativa.

A medida que se adopten nuevas técnicas y herramientas como parte de su cultura, muévase

en el siguiente conjunto de buenas ideas.

Si encuentra que un método que intentó (¡realmente lo intentó!) no agrega valor, no lo haga de nuevo.

Los esfuerzos de mejora del proceso tendrán más éxito si el cultivo permite la experimentación y el respaldo de experimentos fallidos. Las personas que perciben que el riesgo de fracaso es aceptable están más dispuestos a explorar nuevas formas de hacer las cosas. Sin embargo, si los métodos se impusieron desde arriba, sin importar qué tan bien funcionen en sus problemas en su entorno, puede esperar un avance brusco. Nuestro grupo acordó no adoptar ningún dogma metodológico, sino llevar a cabo rutinariamente aquellas actividades de análisis y diseño que hemos aprendido por experiencia nos ayudan a producir mejores sistemas para nuestros clientes.

**Resumen**

Este artículo describe cómo un pequeño grupo de software ha adoptado gradualmente una cultura que proporciona un marco de valores en el que tomar decisiones, establecer prioridades y elegir un camino hacia una mayor actuación.

No puedes comprar una cultura en envoltura retráctil; debes tirar el tuyo.

Cada equipo de software trabaja en un contexto diferente de expectativas, presiones, dominios de aplicaciones y tecnologías.

El proceso de acordar los principios y valores proporciona muchas oportunidades para mejorar tanto el entorno de trabajo como los resultados del trabajo.

 Una cultura compartida es esencial para avanzar a través de la secuencia de madurez del proceso de software hacia la disciplina de procesos de desarrollo de software repetibles y mensurables.

**Reconocimiento**

El mayor determinante de la efectividad del desarrollo de software son los propios miembros del equipo.

Tuve la suerte de comenzar con un equipo de personas que trabajaban bien juntas, se preocupaban por la calidad y estaban dispuestos a probar algunas cosas nuevas. A medida que agregamos nuevos miembros, el equipo siguió mejorando. Mi agradecimiento a todos los que cooperaron conmigo para tratar de mejorar el trabajo que hacemos.