Gestión de proyecto

- Definicion

- Criterios de éxito

1. Entregar el software al cliente en el tiempo acordado.

2. Mantener costos dentro del presupuesto general.

3. Entregar software que cumpla con las expectativas del cliente.

4. Mantener un equipo de desarrollo óptimo y con buen funcionamiento.

- Actividades

1. Planeación del proyecto.

2. Informes.

3. Gestión del riesgo.

4. Gestión de personal.

5. Redactar propuestas.

Gestión de riesgo

- Definicion

- Categorías

1. Riesgos del proyecto.

2. Riesgos del producto.

3. Riesgos empresariales.

- Proceso de Gestión de Riesgo

1. Identificación del riesgo.

2. Análisis de riesgos.

3. Planeación del riesgo.

4. Monitorización del riesgo.

Gestión de personal

- Factores críticos en la gestión de personal

1. Consistencia

2. Respeto

3. Inclusión

4. Honestidad

- Motivación del personal

1. Necesidades Fisiológicas

2. Necesidades de seguridad

3. Necesidades sociales

4. Necesidades de estima

5. Necesidades de autorrealización

- Tipo de Personalidades

1. Personas orientadas a las tareas

2. Personas orientadas hacia sí mismas

3. Personas orientadas a la interacción

Trabajo en equipo

- Beneficios de grupo cohesivo

- Selección de los miembros del grupo

- Organización del grupo

- Comunicaciones grupales

**Gestión de Proyectos**

La gestión de proyectos de software es una parte esencial de la ingeniería de software. Los proyectos necesitan administrarse porque la ingeniería de software está sujeta a restricciones organizacionales de presupuesto y fecha. El trabajo del administrador del proyecto es asegurarse de que el proyecto de software cumpla y supere tales restricciones, además de que entregue software de calidad. La buena gestión no puede garantizar el éxito del proyecto. Sin embargo, la mala gestión da como resultado una falla del proyecto.

Para la mayoría de los proyectos, los criterios de éxito para la gestión del proyecto son:

* Entregar el software al cliente en el tiempo acordado.
* Mantener costos dentro del presupuesto general.
* Entregar software que cumpla con las expectativas del cliente.
* Mantener un equipo de desarrollo óptimo y con buen funcionamiento.

La ingeniería de software es diferente en algunas formas a otros tipos de ingeniería que hacen a la gestión del software particularmente desafiante. Algunas de estas diferencias son:

* **El producto es intangible:** El software es intangible. Los administradores de proyectos de software no pueden constatar el progreso con sólo observar el artefacto que se construye. Más bien, ellos se apoyan en otros para crear la prueba que pueden utilizar al revisar el progreso del trabajo.
* **Los grandes proyectos de software con frecuencia son proyectos excepcionales:** Los grandes proyectos de software se consideran en general diferentes en ciertas formas de los proyectos anteriores. Por eso, incluso los administradores que cuentan con vasta experiencia pueden encontrar difícil anticiparse a los problemas. Los cambios tecnológicos pueden volver obsoleta la experiencia de un administrador.
* **Los procesos de software son variables y específicos de la organización:** Los procesos de software varían considerablemente de una organización a otra. Aunque se ha producido un notorio avance en la estandarización y el mejoramiento de los procesos, no es posible predecir de manera confiable cuándo un proceso de software particular conducirá a problemas de desarrollo.

La labor de un administrador de proyecto de software varía enormemente en función de la organización y el producto de software a desarrollar. No obstante, la mayoría de los administradores toman la responsabilidad de varias o todas las siguientes actividades:

1. **Planeación del proyecto.** Los administradores de proyecto son responsables de la planeación, estimación y calendarización del desarrollo del proyecto, así como de la asignación de tareas a las personas. Supervisan el trabajo para verificar que se realice de acuerdo con los estándares requeridos y monitorean el avance para comprobar que el desarrollo esté a tiempo y dentro del presupuesto.
2. **Informes.** Los administradores de proyectos por lo común son responsables de informar del avance de un proyecto a los clientes y administradores de la compañía que desarrolla el software. Deben ser capaces de comunicarse en varios niveles, desde codificar información técnica detallada hasta labrar resúmenes administrativos. Deben redactar documentos concisos y coherentes que sinteticen información crítica de reportes detallados del proyecto.
3. **Gestión del riesgo.** Los administradores de proyecto tienen que valorar los riesgos que pueden afectar un proyecto, monitorizar dichos riesgos y emprender acciones cuando surjan problemas.
4. **Gestión de personal.** Los administradores de proyecto son responsables de administrar un equipo de personas. Deben elegir a los integrantes de sus equipos y establecer formas de trabajar que conduzcan a desempeño efectivo del equipo.
5. **Redactar propuestas.** La primera etapa en un proyecto de software puede implicar escribir una propuesta para obtener un contrato de trabajo. La propuesta describe los objetivos del proyecto y cómo se realizará. Por lo general, incluye estimaciones de costo y calendarización, además de justificar por qué el contrato del proyecto debería concederse a una organización.

**Gestión del riesgo**

La gestión del riesgo implica anticipar riesgos que pudieran alterar el calendario del proyecto o la calidad del software a entregar, y posteriormente tomar acciones para evitar dichos riesgos. Podemos considerar un riesgo como algo que es preferible que no ocurra. Los riesgos pueden amenazar el proyecto, el software que se desarrolla o a la organización. Por lo tanto, existen tres categorías relacionadas de riesgo:

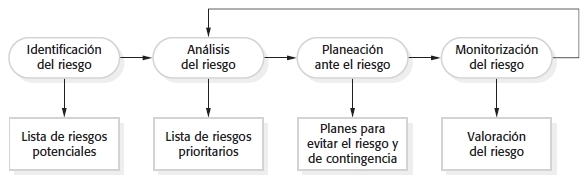
1. **Riesgos del proyecto.** Alteran el calendario o los recursos del proyecto. Un ejemplo de riesgo de proyecto es la renuncia de un diseñador experimentado.
2. **Riesgos del producto.** Afectan la calidad o el rendimiento del software a desarrollar. Un ejemplo de riesgo de producto es la falla que presenta un componente que se adquirió al no desempeñarse como se esperaba.
3. **Riesgos empresariales.** Afectan a la organización que desarrolla o adquiere el software. Por ejemplo, un competidor que introduce un nuevo producto es un riesgo empresarial.

Es necesario registrar los resultados del análisis del riesgo en el plan del proyecto, junto con un análisis de consecuencias, que establece las consecuencias del riesgo para el proyecto, el producto y la empresa. La gestión de riesgos efectiva facilita hacer frente a los problemas y asegurar que éstos no conduzcan a un presupuesto inaceptable o a retrasos en el calendario.

La gestión del riesgo es importante para los proyectos de software debido a la incertidumbre que enfrentan la mayoría de los proyectos. Ésta se deriva de requerimientos vagamente definidos, cambios de requerimientos, dificultades en estimaciones, etc. Es necesario anticipar los riesgos; comprender el efecto de estos riesgos y dar los pasos adecuados para evitar dichos riesgos. Tal vez se necesite diseñar planes de contingencia de manera que se puedan tomar acciones inmediatas de recuperación.

El proceso de gestión del riesgo es un proceso iterativo que continúa a lo largo del proyecto. Una vez desarrollado un plan de gestión del riesgo inicial, se monitoriza la situación para detectar riesgos emergentes. Conforme está disponible más información referente a los riesgos, habrá que volver a analizar los riesgos y decidir si cambió la prioridad del riesgo.

A continuación se ilustra el proceso general de gestión del riesgo y sus etapas:



1. **Identificación del riesgo.**

Se ocupa de identificar los riesgos que pudieran plantear una mayor amenaza al proceso de ingeniería de software, al software a desarrollar, o a la organización que lo desarrolla. La identificación del riesgo puede ser un proceso de equipo, o bien el administrador del proyecto es el que identifica los riesgos más probables. Existen al menos seis tipos de riesgos que se pueden incluirse en una lista de verificación de riesgos:

* Riesgos tecnológicos
* Riesgos personales
* Riesgos organizacionales
* Riesgos de herramientas
* Riesgos de requerimientos
* Riesgos de estimación

Al concluir el proceso de identificación de riesgos, se tendrá una larga lista de eventualidades que podrían ocurrir y afectar al producto, al proceso y a la empresa. Entonces se necesita reducir esta lista a un tamaño razonable. Si existen demasiados riesgos, será prácticamente imposible seguir la huella de todos ellos.

1. **Análisis de riesgos.**

Durante el proceso de análisis de riesgos, hay que considerar cada riesgo identificado y realizar un juicio acerca de la probabilidad y gravedad de dicho riesgo. No es posible hacer valoraciones precisas y numéricas de la probabilidad. En vez de ello, habrá que asignar el riesgo a una de ciertas bandas:

* probabilidad muy baja (< 10%), baja (del 10 al 25%), moderada (del 25 al 50%), alta (del 50 al 75%) o muy alta (> 75%).
* Efectos catastróficos (amenazan la supervivencia del proyecto), graves (causarían grandes demoras), tolerables (demoras dentro de la contingencia permitida) o insignificantes.

Luego hay que tabular los resultados de este proceso de análisis mediante una tabla clasificada de acuerdo con la gravedad. Para hacer una valoración de la probabilidad y seriedad del riesgo se necesita información detallada del proyecto, el proceso, el equipo de desarrollo y la organización.

Una vez analizados y clasificados los riesgos, valore cuáles son los más significativos. Su juicio debe depender de una combinación de la probabilidad de que el riesgo surja junto con los efectos de dicho riesgo. En general, los riesgos catastróficos deben considerarse siempre, así como los riesgos graves con más de una probabilidad moderada de ocurrencia.

Boehm recomienda identificar y monitorizar los 10 riesgos principales, pero considera que esta cifra es más bien arbitraria. El número correcto de riesgos a monitorizar debe depender del proyecto. Sin embargo, el número de riesgos elegidos para monitorizar debe ser manejable.

1. **Planeación del riesgo.**

Considera cada uno de los riesgos clave identificados y desarrolla estrategias para manejarlos. Para cada uno de los riesgos, se consideran las acciones que puede tomar para minimizar la perturbación del proyecto si se produce el problema identificado en el riesgo. También debe pensar en la información que tal vez necesite recopilar mientras observa el proyecto para que pueda anticipar los problemas.

No hay un proceso simple que pueda seguirse para la planeación de contingencias. Se apoya en el juicio y la experiencia del administrador del proyecto. Las posibles estrategias de gestión del riesgo se pueden agrupar en tres categorías:

* **Estrategias de evitación.** Se reducirá la probabilidad de que surja el riesgo.
* **Estrategias de minimización.** Se reducirá el efecto del riesgo.
* **Planes de contingencia.** Se está preparado para lo peor y se tiene una estrategia para hacer frente a ello.

1. **Monitorización del riesgo.**

La monitorización del riesgo es el proceso para comprobar que no han cambiado sus suposiciones sobre riesgos del producto, el proceso y la empresa. Hay que valorar regularmente cada uno de los riesgos identificados para decidir si este riesgo se vuelve más o menos probable. También se tiene que considerar si los efectos del riesgo han cambiado o no. Para hacer esto, observe otros factores, como el número de peticiones de cambio de requerimientos, lo que da pistas acerca de la probabilidad del riesgo y sus efectos. Dichos factores dependen claramente de los tipos de riesgos.

Los riesgos deben monitorizarse comúnmente en todas las etapas del proyecto. En cada revisión administrativa, es necesario reflexionar y estudiar cada uno de los riesgos clave por separado. También hay que decidir si es más o menos probable que surja el riesgo, y si cambiaron la gravedad y las consecuencias del riesgo.

**Gestión de personal**

Las personas que trabajan en una organización de software son los activos más importantes. Cuesta mucho dinero reclutar y retener al buen personal, así que depende de los administradores de software garantizar que la organización obtenga el mejor aprovechamiento posible por su inversión. En las compañías y economías exitosas, esto se logra cuando la organización respeta a las personas y les asigna responsabilidades que reflejan sus habilidades y experiencia.

Es importante que los administradores de proyecto de software comprendan los conflictos técnicos que influyen en el trabajo del desarrollo de software. Sin embargo, los buenos ingenieros de software no necesariamente son buenos administradores de personal, pueden carecer de habilidades más sutiles que les permitan motivar y dirigir a un equipo de desarrollo de proyecto. Como administrador de proyecto, usted deberá estar al tanto de los problemas potenciales de administrar personal y debe tratar de desarrollar habilidades de gestión de recursos humanos.

Desde la perspectiva del autor, existen cuatro factores críticos en la gestión de personal:

1. **Consistencia** Todas las personas en un equipo de proyecto deben recibir un trato similar, las personas podrían sentir que sus aportaciones a la organización se menosprecian.
2. **Respeto** Las personas tienen distintas habilidades y los administradores deben respetar esas diferencias. Todos los miembros del equipo deben recibir una oportunidad para aportar.
3. **Inclusión** Las personas contribuyen efectivamente cuando sienten que otros las escuchan y que sus propuestas se toman en cuenta. Es importante desarrollar un ambiente laboral donde se consideren todas las visiones, incluso las del personal más joven.
4. **Honestidad** Como administrador, siempre debe ser honesto acerca de lo que está bien y lo que está mal en el equipo. También debe ser honesto respecto a su nivel de conocimiento técnico y voluntad para comunicar al personal más conocimiento cuando sea necesario. Si trata de encubrir la ignorancia o los problemas, con el tiempo, éstos saldrán a la luz y perderá el respeto del grupo.

**Motivación del personal**

Como administrador de proyecto, usted necesitará motivar a las personas con quienes trabaja, de manera que éstas contribuyan con lo mejor de sus habilidades. Motivación significa organizar el trabajo y el ambiente laboral para alentar a los individuos a desempeñarse tan efectivamente como sea posible. Si las personas no están motivadas, no estarán interesadas en la actividad que realizan. Así que trabajarán con lentitud, y será más probable que cometan errores y que no contribuyan con las metas más amplias del equipo o la organización.

Para fomentar este ánimo, hay que saber un poco acerca de qué motiva a la gente. Maslow (1954) sugiere que las personas se sienten motivadas para cubrir sus necesidades, las cuales se ordenan en una serie de niveles. Los niveles más bajos de esta jerarquía representan necesidades fundamentales de alimentación, sueño, etcétera, y la necesidad de sentirse seguro en un ambiente. Las necesidades sociales se relacionan con el hecho de sentirse parte de un grupo social. Las necesidades de estima representan la necesidad de sentirse respetado por otros, y las necesidades de autorrealización tienen que ver con el desarrollo personal. Las personas requieren cubrir las necesidades de nivel inferior, como el hambre, antes de las necesidades de nivel superior, que son más abstractas.



Las personas que trabajan en organizaciones de desarrollo de software, por lo general, no están hambrientas ni sedientas ni físicamente amenazadas por su ambiente. Por lo tanto, asegurarse de que se cubren las necesidades sociales, de estima y autorrealización de las personas es más importante desde un punto de vista administrativo.

1. Para satisfacer las necesidades sociales, es preciso dar a las personas tiempo para reunirse con sus compañeros de trabajo y proporcionarles lugares de socialización. Esto es relativamente sencillo cuando todos los miembros de un equipo de desarrollo trabajan en el mismo lugar. Aunque cada vez es más común que los miembros del equipo no laboren en el mismo edificio o ni siquiera en la misma ciudad o estado. Los sistemas de redes sociales y las teleconferencias facilitan las comunicaciones, pero los sistemas electrónicos son más efectivos una vez que las personas se conocen entre sí. Por lo tanto, es necesario programar algunas reuniones cara a cara en etapas tempranas del proyecto, de manera que la gente pueda interactuar directamente con otros miembros del equipo. Mediante esta interacción directa, las personas se vuelven parte de un grupo social y aceptan las metas y prioridades de dicho grupo.
2. Para cubrir las necesidades de estima, es necesario demostrar a las personas que son valoradas por la organización. El reconocimiento público de los logros es una forma simple, aunque efectiva, de hacer esto. Desde luego, las personas también deben sentir que se les paga de acuerdo con sus habilidades y experiencia.
3. Finalmente, para cubrir las necesidades de autorrealización, es necesario dar responsabilidad a las personas por su trabajo, asignarles tareas demandantes y ofrecer un programa de capacitación donde puedan desarrollar sus habilidades. La capacitación es una importante influencia motivadora, pues la mayoría de las personas desean adquirir nuevos conocimientos y aprender nuevas habilidades.

El modelo de motivación de Maslow es útil sólo hasta cierto punto, ya que adopta un punto de vista exclusivamente personal de la motivación. No considera de manera adecuada el hecho de que las personas se sienten parte de una organización, un grupo profesional y una o más culturas. Ésta no es tan sólo una cuestión de cubrir necesidades sociales: las personas pueden sentirse motivadas al ayudar a un grupo a lograr metas compartidas.

Ser miembro de un grupo cohesivo es enormemente motivador para la mayoría de la gente. Con frecuencia, las personas con trabajos satisfactorios disfrutan ir a trabajar, porque están motivadas por la gente con la que trabajan y por la actividad que realizan. Por lo tanto, además de pensar en la motivación individual, también hay que considerar cómo un grupo en su conjunto puede motivarse para lograr las metas de la organización.

El tipo de personalidad también influye en la motivación. Bass y Dunteman (1963) clasifican a los profesionales en tres tipos:

* **Personas orientadas a las tareas**, se trata de personas que están motivadas por el reto intelectual de desarrollar software.
* **Personas orientadas hacia sí mismas**, quienes están motivadas principalmente por el éxito y el reconocimiento personales. Están interesadas en el desarrollo del software como un medio para lograr sus propias metas a plazos más largos, como el avance profesional.
* **Personas orientadas a la interacción**, quienes están motivadas por la presencia y las acciones de los compañeros de trabajo. Conforme el desarrollo de software se vuelve más centrado en el usuario, los individuos orientados a la interacción se involucran más en la ingeniería de software.

La motivación de cada individuo se constituye con elementos de cada clase, pero, por lo regular, un tipo de motivación domina en algún momento dado. Sin embargo, los individuos pueden cambiar.

**Trabajo en equipo**

La mayor parte del software profesional se desarrolla mediante equipos de proyecto, cuyo número de miembros varía entre dos y varios cientos de personas. Sin embargo, como es imposible que todos los integrantes de un grupo grande trabajen en conjunto en un solo problema, los equipos grandes habitualmente se dividen en grupos más pequeños. Cada grupo es responsable de desarrollar parte del sistema global. Como regla general, los grupos del proyecto de ingeniería de software no deben tener más de 10 miembros. Cuando se usan grupos pequeños se reducen los problemas de comunicación. Todos conocen a todos los demás, y el grupo en su conjunto puede reunirse en torno a una mesa para estudiar el proyecto y el software que desarrollan.

Conformar un grupo que tiene el equilibrio justo de habilidades técnicas, experiencia y personalidades es una tarea administrativa fundamental. Sin embargo, los grupos exitosos son mucho más que una colección de individuos con el equilibrio justo de habilidades. Un buen equipo es cohesivo y tiene espíritu de grupo. Las personas que participan están motivadas tanto por el éxito del grupo como por sus metas personales.

En un grupo cohesivo, los miembros piensan que el equipo es más importante que los individuos que lo integran. Los miembros de un grupo cohesivo bien liderado son leales al equipo. Se identifican con las metas del grupo y con los demás miembros. Tratan de proteger al grupo, como entidad, de cualquier interferencia externa. Esto hace que el grupo sea sólido y pueda enfrentar problemas y situaciones inesperadas.

Los beneficios de crear un grupo cohesivo son:

1. **El grupo puede establecer sus propios estándares de calidad** Puesto que dichos estándares se establecen por consenso, éstos tienen más probabilidad de respetarse que los estándares externos impuestos sobre el grupo.
2. **Los individuos aprenden de los demás y se apoyan mutuamente** Las personas en el grupo aprenden de los demás. Las inhibiciones causadas por la ignorancia se minimizan mientras se promueve el aprendizaje mutuo.
3. **El conocimiento se comparte** Puede mantenerse la continuidad si sale un miembro del grupo. Otros en el grupo pueden tomar el control de las tareas críticas para asegurar que el proyecto no se altere en forma considerable.
4. **Se alientan la refactorización y el mejoramiento continuo** Los miembros del grupo trabajan de manera colectiva para entregar resultados de alta calidad y corregir problemas, sin importar quiénes crearon originalmente el diseño o programa.

Los buenos administradores de proyecto siempre tratan de alentar la cohesión grupal. Pueden organizar eventos sociales para los miembros del grupo y sus familias, además de establecer un sentido de identidad al ponerle nombre al grupo y establecer una compatibilidad y un territorio grupales, u organizar actividades explícitas de construcción grupal, como actividades deportivas y juegos.

Una de las formas más efectivas de fomentar la cohesión es ser comprensivo. Esto significa que hay que tratar a los miembros de los grupos como responsables y confiables, y poner la información a libre disposición. En ocasiones, los administradores sienten que no pueden revelar cierta información a todos los miembros del grupo. Esto invariablemente crea un clima de desconfianza. El simple intercambio de información es una forma efectiva de hacer que las personas se sientan valoradas y se reconozcan como parte de un grupo.

Si un grupo es efectivo o no, en cierta medida, depende de la naturaleza del proyecto y la organización que realiza el trabajo. Si una organización se encuentra en un estado de turbulencia por reorganizaciones constantes e inseguridad laboral, es muy difícil que los miembros del equipo se enfoquen en el desarrollo de software. Sin embargo, aparte de los conflictos del proyecto y la organización, existen tres factores genéricos que afectan el trabajo en equipo:

1. **Las personas en el grupo** Se necesita una combinación de personas en un grupo de proyecto, puesto que el desarrollo de software implica diversas actividades, como negociación con clientes, programación, pruebas y documentación.
2. **La organización grupal** Un grupo debe organizarse de forma que los individuos puedan contribuir con sus mejores habilidades y completar las tareas como se esperaba.
3. **Comunicaciones técnicas y administrativas** Es esencial la óptima comunicación entre los miembros del grupo, y entre el equipo de ingeniería de software y otras partes interesadas en el proyecto.

Como en todos los conflictos administrativos, reunir al equipo correcto no garantiza el éxito del proyecto. Muchas otras cosas pueden salir mal, incluidos los cambios en los negocios y en el ambiente empresarial. Sin embargo, si no se presta la atención debida a la composición, la organización y las comunicaciones del grupo, aumenta la probabilidad de que el proyecto enfrente dificultades.

**Selección de los miembros del grupo**

La labor de un administrador o líder de equipo es crear un grupo cohesivo y organizar a los miembros del grupo para que puedan trabajar en conjunto de manera efectiva. Esto implica crear un grupo con el equilibrio correcto de habilidades técnicas y personalidades, así como organizarlo para que los miembros trabajen adecuadamente en conjunto. En ocasiones, se contrata a personas externas a la organización; no obstante, con más frecuencia, los grupos de ingeniería de software se componen de empleados actuales que tienen experiencia adquirida en otros proyectos. Con todo, los administradores pocas veces tienen absoluta libertad en la selección del equipo. Con frecuencia deben recurrir a las personas que estén disponibles en la compañía, aun cuando no sean ideales para el puesto.

Muchos ingenieros de software están motivados principalmente por su trabajo. Por lo tanto, los grupos de desarrollo a menudo están compuestos por personas que cuentan con ideas propias sobre qué problemas técnicos deben resolverse. Esto se refleja en los problemas que se reportan regularmente relacionados con estándares de interfaz ignorados, sistemas rediseñados conforme se codifican, arreglo innecesario del sistema, etcétera.

Un grupo con personalidades complementarias puede trabajar mejor que un grupo seleccionado exclusivamente por la habilidad técnica. Es probable que las personas que están motivadas por el trabajo sean las más fuertes técnicamente. Las personas que son orientadas hacia sí mismas tal vez serán mejores para impulsar el trabajo hacia delante para terminar la tarea. Las personas orientadas a la interacción ayudan a facilitar las comunicaciones dentro del grupo. Se considera que en el grupo es particularmente importante contar con personas orientadas a la interacción. A éstas les gusta hablar con los demás y son capaces de detectar tensiones y diferencias en una etapa temprana, antes de que tengan serias repercusiones sobre el grupo.

En ocasiones es imposible elegir un grupo con personalidades complementarias. Si éste es el caso, el administrador del proyecto tiene que controlar al grupo de modo que las metas individuales no se antepongan a los objetivos de la organización y del grupo. Este control es más sencillo de lograr si todos los miembros del grupo participan en cada etapa del proyecto. La iniciativa individual es más factible cuando los miembros del grupo reciben instrucciones sin estar al tanto de la parte que desempeña su tarea en el proyecto global.

**Organización del grupo**

La forma en que se organiza un grupo influye en las decisiones que toma dicho grupo, las maneras como se intercambia la información y las interacciones entre el grupo de desarrollo y los participantes externos del proyecto. Las preguntas organizacionales importantes para los administradores de proyecto incluyen:

1. **¿El administrador del proyecto debe ser el líder técnico del grupo?** El líder técnico o arquitecto del sistema es responsable de las decisiones técnicas críticas tomadas durante el desarrollo del software. En ocasiones, el administrador del proyecto tiene la habilidad y experiencia para desempeñar este papel. Sin embargo, en caso de grandes proyectos, es mejor asignar a un ingeniero con experiencia como el arquitecto del proyecto, quien tomará la responsabilidad del liderazgo técnico.
2. **¿Quién se encargará de tomar las decisiones técnicas críticas, y cómo se tomarán? ¿Las decisiones las tomará el arquitecto del sistema, el administrador del proyecto o se llegará a un consenso entre un rango más amplio de miembros del equipo?**
3. **¿Cómo se manejarán las interacciones con los participantes externos y los altos directivos de la compañía?** En muchos casos, el administrador del proyecto será el responsable de dichas interacciones, asistido por el arquitecto del sistema. Sin embargo, un modelo de organización alternativo incluye una función exclusiva para las relaciones externas, lo que supone asignar para dicha función a una persona con habilidades de interacción adecuadas.
4. **¿Cómo es posible que los grupos logren integrar a personas que no se localizan en el mismo lugar?** Ahora es común que los grupos incluyan a miembros de diferentes organizaciones y personas que trabajan desde casa o en oficinas compartidas. Esto debe tomarse en cuenta en los procesos de toma de decisiones grupales.
5. **¿Cómo puede compartirse el conocimiento a través del grupo?** La organización del grupo afecta el intercambio de información, pues determinadas formas de organización son mejores que otras para compartir. Sin embargo, conviene evitar demasiado intercambio de información, ya que las personas pueden sobresaturarse y la información excesiva podría distraerlos de sus labores.

Los grupos de programación pequeños, por lo general, están organizados en una forma bastante informal. El líder del grupo participa en el desarrollo de software con los otros miembros del grupo. En un grupo informal, todo el equipo analiza el trabajo a realizar, y las tareas se asignan según la habilidad y la experiencia. Los miembros del grupo con mayor jerarquía pueden ser responsables del diseño arquitectónico. No obstante, el diseño y la implementación detallados son compromisos del miembro del equipo que se asigna a una tarea particular.

Los grupos informales pueden ser muy exitosos, en particular cuando la mayoría de los miembros del grupo son experimentados y competentes. Tal grupo toma decisiones por consenso, lo que mejora la cohesión y el rendimiento. Sin embargo, si un grupo está compuesto principalmente por miembros inexpertos e incompetentes, la informalidad puede ser un obstáculo porque no existe autoridad definida para dirigir el trabajo, lo que causa una falta de coordinación entre los miembros del grupo y, posiblemente, una eventual falla del proyecto.

Los grupos jerárquicos son grupos que comparten una estructura jerárquica con el líder del grupo en la parte superior del escalafón. El líder tiene autoridad más formal que los miembros del grupo y así puede dirigir el trabajo. Existe una clara estructura organizacional, y las decisiones se toman hacia la parte superior de la jerarquía y se aplican por las personas que están más abajo en la jerarquía. Las comunicaciones, ante todo, son instrucciones del personal ejecutivo y existe relativamente poca comunicación ascendente, es decir, desde los niveles más bajos hacia los niveles superiores en la jerarquía.

Este enfoque funciona bien cuando un problema bien entendido puede descomponerse fácilmente en subproblemas en los que las soluciones se desarrollan en diferentes partes de la jerarquía. En dichas situaciones se requiere muy poca comunicación a través de la jerarquía. Sin embargo, tales situaciones, en proporción, son poco comunes en la ingeniería de software por las siguientes razones:

1. Los cambios al software requieren con frecuencia cambios en varias partes del sistema y esto conduce a una discusión y negociación en todos los niveles de la jerarquía.
2. Las tecnologías de software cambian tan rápido que muchas veces el personal más joven conoce más de la tecnología que el personal experimentado. Las comunicaciones descendentes pueden significar que el administrador del proyecto no vislumbra las oportunidades de usar nuevas tecnologías. El personal más joven puede frustrarse debido a que considera obsoletas las tecnologías usadas para el desarrollo.

Las organizaciones grupales democráticas y jerárquicas no reconocen formalmente que puede haber diferencias muy grandes de habilidad técnica entre los miembros del grupo. Los mejores programadores pueden ser hasta 25 veces más productivos que los peores programadores. Tiene sentido aprovechar las capacidades de los mejores elementos en la forma más efectiva y brindarles tanto apoyo como sea posible. Uno de los primeros modelos organizacionales que tenía la intención de ofrecer apoyo fue el llamado equipo programador jefe.

**Comunicaciones grupales**

Es absolutamente esencial que los miembros del grupo se comuniquen efectiva y eficientemente entre sí y con otras partes interesadas en el proyecto. Los miembros del grupo deben intercambiar información acerca del estatus de su trabajo, las decisiones de diseño que se tomaron y los cambios a las decisiones de diseño previas. Tienen que resolver los problemas que surjan con otros interesados en el proyecto e informar a éstos sobre los cambios al sistema, grupo y planes de entrega. La buena comunicación ayuda también a fortalecer la cohesión del grupo. Los miembros del grupo llegan a entender las motivaciones, fortalezas y debilidades de otras personas en el grupo.

La efectividad y la eficiencia de las comunicaciones están influidas por:

1. **Tamaño del grupo.** Conforme el grupo crece, se hace más difícil que los miembros se comuniquen de manera efectiva. Las diferencias de estatus entre los miembros del grupo significan que las comunicaciones con frecuencia son unidireccionales.
2. **Estructura del grupo.** Los grupos estructurados informalmente se comunican más efectivamente que los grupos con una estructura jerárquica formal. En los grupos jerárquicos, las comunicaciones tienden a fluir hacia arriba y abajo de la jerarquía. Las personas en el mismo nivel tal vez no se comuniquen entre sí.
3. **Composición del grupo.** Las personas con los mismos tipos de personalidad pueden chocar y, como resultado, las comunicaciones se inhiben. Además, la comunicación es mejor en los grupos integrados por personas de uno y otro género que en los grupos formados por miembros de un solo género.
4. **El ambiente laboral físico.** La organización del centro de trabajo es un factor importante para facilitar o inhibir las comunicaciones.
5. **Los canales de comunicación disponibles.** Conforme los equipos de proyecto se distribuyen cada vez más, con miembros de equipo que trabajan en lugares remotos, es necesario utilizar varias tecnologías para facilitar las comunicaciones.

Los administradores de proyecto trabajan bajo plazos estrechos y tratan de usar canales de comunicación que no consuman mucho tiempo. Por lo tanto, se apoyan en reuniones y documentos formales para transmitir la información al personal del proyecto y las partes interesadas. Aunque éste tal vez sea un enfoque eficiente para la comunicación desde la perspectiva de un administrador de proyecto, comúnmente no es muy efectivo. A menudo existen buenas razones por las que las personas no pueden asistir a las reuniones y, por lo tanto, no escuchan la presentación. Los documentos extensos casi nunca se leen, debido a que los lectores no saben si los documentos son relevantes. Al producirse varias versiones del mismo documento, los lectores encuentran difícil hacer un seguimiento de los cambios.

La comunicación efectiva se logra cuando las comunicaciones son bidireccionales, y las personas implicadas pueden discutir los conflictos y la información, y establecer una comprensión común de las proposiciones y los problemas. Esto se logra mediante reuniones, aunque éstas suelen estar dominadas por las personalidades poderosas. En ocasiones no es práctico citar con escasa anticipación a reuniones. Cada vez más equipos de proyecto incluyen miembros ubicados a distancia, lo que dificulta las reuniones.

Para contrarrestar estos problemas y apoyar el intercambio de información, se puede recurrir a tecnologías Web, como wikis y blogs. Los wikis respaldan la creación y edición de documentos en colaboración, mientras que los blogs apoyan las discusiones generadas por preguntas y comentarios hechos por los miembros del grupo. Los wikis y blogs permiten a los miembros del proyecto y a los participantes externos intercambiar información, sin importar su ubicación. Ayudan a gestionar la información y a seguir la huella de los hilos de discusión. Para resolver conflictos que necesiten discusión, se puede recurrir a la mensajería instantánea y las teleconferencias, las cuales pueden organizarse fácilmente.

Planeación de proyectos

- Definicion

- Momentos para la planeación

1. En la etapa de propuestas

2. Durante la fase de inicio

3. Periódicamente a lo largo del proyecto

- Parámetros para calcular costos de un proyecto

1. costos de esfuerzo

2. costos de hardware y software

3. costos de viajes y capacitación

- Definir mecanismos de monitorización

- El plan evoluciona durante el desarrollo

Fijación de precio al Software

Desarrollo dirigido por un plan

- Definición

- Pros / Contras

- Planes de Proyecto

1. Introducción

2. Organización del proyecto

3. Análisis de riesgo

4. Requerimientos de recursos de hardware y software

5. División del trabajo

6. Calendario del proyecto

7. Mecanismos de monitorización y reporte

- El proceso de planeación

Calendarización de Proyectos

- Definición

- Representación del calendario

1. Gráficas de barras, basadas en el calendario (Gantt)

2. Redes de actividad

- Componentes de una actividad

1. duración

2. estimación del esfuerzo

3. plazo

4. punto final definido

Planeación ágil

Técnicas de Estimación

- Tipos de técnicas

1. basada en la experiencia

2. modelado algorítmico de costo

- El modelo COCOMO II

1. modelo de composición de aplicación

2. modelo de diseño temprano

3. modelo de reutilización

4. modelo pos-arquitectónico

**Planeación de proyectos**

La planeación de proyectos es una de las labores más importantes de un administrador de proyectos de software. Como administrador, debe dividir el trabajo en partes y asignar éstas a los miembros del equipo del proyecto, anticipar los problemas que pudieran surgir y preparar posibles soluciones a tales inconvenientes. El plan creado al comienzo de un proyecto se usa para comunicar al equipo y los clientes cómo se realizará el trabajo, así como para ayudar a valorar el avance del proyecto.

La planeación se presenta durante tres etapas en un ciclo de vida del proyecto:

1. **En la etapa de propuestas**, cuando se presenta una licitación con vistas a obtener un contrato para desarrollar o proporcionar un sistema de software. En esta etapa es necesario un plan para ayudarle a decidir si cuenta con los recursos para completar el trabajo y a calcular el precio que debe cotizar al cliente.
2. **Durante la fase de inicio**, cuando debe determinar quién trabajará en el proyecto, cómo se dividirá el proyecto en incrementos, cómo se asignarán los recursos a través de su compañía, etc.
3. **Periódicamente a lo largo del proyecto**, cuando el plan se modifica a la luz de la experiencia obtenida y la información del monitoreo del avance del trabajo. Se aprende más acerca del sistema a implementar y de las capacidades del equipo de desarrollo. Esta información permite hacer estimaciones más precisas sobre cuánto tardará el trabajo. Más aún, es probable que los requerimientos del software cambien y esto, por lo general, significa que debe modificarse la división del trabajo y extenderse el plazo.

La planeación en la etapa de propuesta es especulativa, pues muchas veces no se cuenta con un conjunto completo de requerimientos para el software a desarrollar. En vez de ello, hay que responder a una solicitud de propuestas basada en una descripción de alto nivel sobre la funcionalidad del software que se requiere.

Cuando se presenta una licitación para obtener un contrato, hay que calcular el precio que se propondrá al cliente para el desarrollo del software. Como punto de partida para calcular este precio, se requiere presentar una estimación de los costos para completar el trabajo del proyecto. La estimación incluye calcular cuánto esfuerzo se requiere para terminar cada actividad y calcular el costo total de las actividades. Una vez que se tiene una estimación razonable de los probables costos, entonces es posible calcular el precio que se cotizará al cliente.

Existen tres principales parámetros que se deben usar al calcular los costos de un proyecto de desarrollo de software:

* **costos de esfuerzo** (los costos de pagar a los ingenieros y administradores de software)
* **costos de hardware y software**
* **costos de viajes y capacitación**

Para la mayoría de los proyectos, el mayor costo es el esfuerzo. Debe estimarse el esfuerzo total (en meses-hombre) que es probable se requiera para completar el trabajo de un proyecto. Desde luego, se cuenta con datos limitados para realizar tal valoración, de manera que habrá que hacer la mejor evaluación posible y a continuación agregar contingencia significativa en caso de que la estimación inicial sea optimista.

Para sistemas comerciales a menudo se usa hardware commodity, que es relativamente barato. Sin embargo, los costos de software pueden ser considerables si se debe licenciar el middleware y el software de plataforma. Es posible que se requieran viajes frecuentes cuando un proyecto se desarrolla en diferentes lugares.

Una vez asignado un contrato para desarrollar un sistema, debe afinarse un bosquejo de plan de proyecto para crear un plan de inicio de proyecto. En esta etapa se debe saber más acerca de los requerimientos para este sistema. Sin embargo, es posible que no se cuente con una especificación completa de los requerimientos. Su meta durante esta etapa debe ser elaborar un plan de proyecto que pueda usarse para apoyar la toma de decisiones acerca del personal y el presupuesto del proyecto. El plan sirve como base para asignar recursos al proyecto desde el interior de la organización y decidir si se requiere la contratación de nuevo personal.

El plan debe definir también los mecanismos de monitorización del proyecto. Es necesario hacer un seguimiento del avance del proyecto y comparar los avances y costos reales con el progreso planeado. Un buen administrador también debe ser capaz de crear una imagen clara de lo que sucede mediante comunicaciones informales con el personal del proyecto. La monitorización informal ayuda a pronosticar problemas potenciales de proyecto al revelar las dificultades conforme ocurren.

El plan del proyecto siempre evoluciona durante el proceso de desarrollo. La planeación del desarrollo pretende garantizar que el plan del proyecto siga siendo un documento útil para que el personal comprenda lo que debe lograrse y cuándo debe entregarse. Por lo tanto, el calendario y la estimación de costos y de riesgos deben revisarse a medida que se desarrolla el software.

**Fijación de precio al software**

En principio, el precio de un producto de software a un cliente es simplemente el costo del desarrollo más las ganancias para el diseñador. Sin embargo, en la práctica, la relación entre el costo del proyecto y el precio cotizado al cliente no es tan simple. Cuando se calcula un precio, hay que hacer consideraciones más amplias de índole organizacional, económica, política y empresarial. Debe pensarse en los intereses de la empresa, los riesgos asociados con el proyecto y el tipo de contrato que se firmará. Esto puede hacer que el precio se ajuste al alza o baja. Dadas las precisiones organizacionales implicadas, decidir sobre un precio de proyecto debe ser una actividad grupal que incluya al personal de marketing y ventas, así como altos ejecutivos y administradores de proyecto.

Como el costo de un proyecto sólo está débilmente relacionado con el precio cotizado a un cliente, “cotizar para ganar” es una estrategia usada comúnmente. Cotizar para ganar significa que una compañía tiene alguna idea del precio que el cliente espera pagar y hace una apuesta por el contrato con base en el precio esperado por el cliente. Esto puede parecer no ético y poco práctico en los negocios, pero tiene ventajas tanto para el cliente como para el proveedor del sistema.

Un costo de proyecto se acuerda sobre la base de un borrador de propuesta. Entonces las negociaciones tienen lugar entre cliente y consumidor para establecer la especificación detallada del proyecto. A esta especificación la restringe el costo acordado. El comprador y el vendedor deben convenir cuál es la funcionalidad aceptable del sistema. En muchos proyectos el factor fijo no está constituido por los requerimientos del proyecto, sino por el costo. Los requerimientos pueden cambiar para que el costo no se supere.

**Desarrollo dirigido por un plan**

El desarrollo dirigido por un plan es un enfoque para la ingeniería de software donde el proceso de desarrollo se planea a detalle. Se elabora un plan de proyecto que registra el trabajo que se va a realizar, quién lo efectuará, el calendario de desarrollo y los productos de trabajo. Los administradores utilizan el plan para apoyar la toma de decisiones del proyecto y también como una forma de medir el progreso. El desarrollo dirigido por un plan se sustenta en técnicas de administración de proyectos de ingeniería y pueden considerarse como la manera tradicional de administrar grandes proyectos de desarrollo de software. Lo anterior contrasta con el desarrollo ágil, en el que muchas decisiones que afectan el desarrollo se retrasan y se hacen posteriormente, según se requiera, durante el proceso de desarrollo.

El principal argumento contra el desarrollo basado en un plan es que muchas decisiones tempranas deben revisarse debido a cambios al entorno en los que se desarrollará y usará el software. Retrasar las decisiones es una práctica sensata porque evita tener que volver a trabajar.

Los argumentos en favor de un enfoque dirigido por un plan son que la planeación temprana permite que los asuntos de la organización (disponibilidad de personal, otros proyectos, etcétera) se tomen estrictamente en cuenta, y que los problemas potenciales y dependencias se descubran antes de que se inicie el proyecto, y no cuando ya esté en marcha.

Desde esta perspectiva, el mejor enfoque a la planeación del proyecto incluye una mezcla juiciosa de desarrollo basado en un plan y ágil. El equilibrio depende del tipo de proyecto y de las habilidades del personal que estén disponibles. Por un lado, los grandes sistemas críticos de seguridad y protección requieren un amplio análisis previo y quizá certificarse antes de utilizarse. Esto debe estar dirigido mediante un plan. Por el otro lado, los sistemas de información que se usan en un entorno competitivo vertiginosamente cambiante, deben ser ágiles. Cuando muchas compañías están implicadas en un proyecto de desarrollo, con frecuencia se usa un enfoque basado en un plan para coordinar el trabajo a través de cada sitio de desarrollo.

**Planes de proyecto**

En un proyecto de desarrollo dirigido por un plan, en el plan de proyecto se establecen los recursos disponibles para el proyecto, la división del trabajo y un calendario para realizar el trabajo. El plan debe identificar los riesgos para el proyecto y el software en desarrollo, así como el enfoque que se toma para la gestión del riesgo. Los planes incluyen por lo regular las siguientes secciones:

1. **Introducción.** Ésta describe brevemente los objetivos del proyecto y establece las restricciones que afectan la administración del proyecto.
2. **Organización del proyecto.** Ésta refiere la forma en que está organizado el equipo de desarrollo, las personas implicadas y sus roles en el equipo.
3. **Análisis de riesgo.** Detalla los posibles riesgos del proyecto, la probabilidad de que surjan dichos riesgos y las estrategias propuestas para reducir el riesgo.
4. **Requerimientos de recursos de hardware y software.** Detallan el hardware y el software de soporte requeridos para realizar el desarrollo.
5. **División del trabajo.** Establece la división del proyecto en actividades e identifica los plazos y las entregas asociados con cada actividad. Los plazos son las etapas clave del proyecto donde puede valorarse el avance; las entregas son productos de trabajo que se proporcionan al cliente.
6. **Calendario del proyecto.** Indica las dependencias entre las actividades, el tiempo estimado requerido para alcanzar cada plazo y la asignación de personal a las actividades.
7. **Mecanismos de monitorización y reporte.** Esta sección define los informes administrativos que deben producirse, cuando tienen que elaborarse y los mecanismos de monitorización del proyecto que se usarán.

**El proceso de planeación**

La planeación del proyecto es un proceso iterativo que comienza cuando se diseña un plan de proyecto inicial durante la fase de arranque del proyecto. Los cambios al plan son inevitables. Conforme más información sobre el sistema y el equipo esté disponible durante el proyecto, habrá que revisar regularmente el plan para reflejar los requerimientos, el calendario y los cambios en el riesgo. Modificar las metas de la empresa conduce también a cambios en los planes del proyecto. A medida que cambien las metas de la empresa, esto podría afectar a todos los proyectos, los cuales tal vez deban replantearse.

Al comienzo de un proceso de planeación, hay que valorar las restricciones que afectan el proyecto. Éstas son fecha de entrega requerida, personal disponible, presupuesto global, herramientas disponibles, etcétera. En conjunción con esto, también hay que identificar los hitos y entregables del proyecto. Los hitos son puntos en el calendario contra los que puede valorar el avance. Los entregables son productos de trabajo que se proporcionan al cliente.

Entonces el proceso entra en un ciclo. Se prepara un calendario estimado para el proyecto y se inician las actividades definidas en el calendario o se concede el permiso para continuarlas. Después de cierto tiempo, se debe revisar el avance y anotar las diferencias del calendario planeado. Puesto que las estimaciones iniciales de los parámetros del proyecto inevitablemente son aproximadas, es normal que se presenten atrasos menores y habrá que hacer modificaciones al plan original.

Es importante ser realista al elaborar un plan de proyecto. Los problemas de alguna descripción surgen casi siempre durante un proyecto y pueden conducir a demoras del mismo. En consecuencia, las suposiciones y la calendarización iniciales deben ser más pesimistas que optimistas. Tiene que haber suficiente contingencia acumulada en el plan, de modo que las restricciones y los hitos del plan no necesiten renegociarse cada vez que se revisa el ciclo de planeación.

Si existen graves problemas con el trabajo de desarrollo que conduzcan a demoras significativas, habrá que iniciar acciones de mitigación del riesgo para reducir los riesgos de falla del proyecto. Junto con dichas acciones, se debe también replantear el proyecto. Esto puede incluir renegociar las restricciones del proyecto y entregables con el cliente. También es necesario establecer y acordar con el cliente un nuevo calendario sobre el tiempo en que se completará el trabajo.

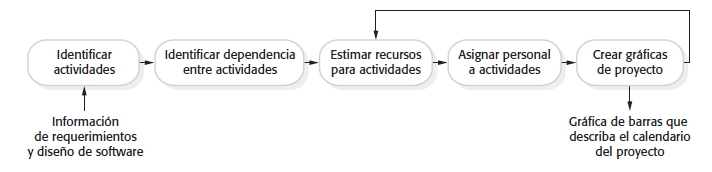
Si esta renegociación no tiene éxito o si no son efectivas las acciones de mitigación del riesgo, se debe organizar entonces una revisión técnica formal del proyecto. Los objetivos de esta revisión son encontrar un enfoque alternativo que permita la continuación del proyecto, y comprobar si éste, así como las metas del cliente y el desarrollador de software, todavía están alineados.

El resultado de una revisión puede ser una decisión para cancelar un proyecto. Esto podría obedecer a un efecto de los fracasos técnicos o administrativos pero, a menudo, es consecuencia de cambios externos que afectan al proyecto. Durante este tiempo, los objetivos y las prioridades de la compañía cambian inevitablemente. Tales cambios pueden significar que el software ya no se requiere más o que los requerimientos del proyecto original resultan inadecuados. Entonces la administración puede decidir detener el desarrollo del software o realizar grandes cambios al proyecto para que éstos reflejen los cambios en los objetivos de la organización.

**Calendarización de proyectos**

La calendarización de proyectos es el proceso de decidir cómo se organizará el trabajo en un proyecto como tareas separadas, y cuándo y cómo se ejecutarán dichas tareas. Se estima el tiempo calendario para completar cada tarea, el esfuerzo requerido y quién trabajará en las tareas identificadas. También hay que estimar los recursos necesarios para completar cada tarea, el tiempo que se necesitará el hardware especializado y cuál será el presupuesto de viajes. Un calendario de proyecto inicial se elabora por lo general durante la fase de arranque del proyecto. Luego, durante la planeación del desarrollo, este calendario se afina y modifica.

Tanto los procesos basados en un plan como los ágiles precisan de un calendario de proyecto inicial. Este calendario inicial se utiliza para planear cómo se asignarán las personas al proyecto y comprobar el avance de éste frente a los compromisos contractuales. En los procesos tradicionales de desarrollo, el calendario completo se elabora inicialmente y enseguida se modifica conforme avanza el proyecto. En los procesos ágiles debe existir un calendario global que identifique el tiempo en que se completarán las principales fases del proyecto. Entonces, se usa un enfoque iterativo de calendarización para planear cada fase.



La calendarización en los proyectos dirigidos por un plan implica dividir el trabajo total de un proyecto en tareas separadas y estimar el tiempo requerido para completar cada tarea. Por lo general, las tareas deben durar al menos una semana, pero no más de dos meses. Una subdivisión más fina significa que una cantidad desproporcionada de tiempo debe emplearse para volver a planear y actualizar el plan del proyecto. La cantidad máxima de tiempo para cualquier tarea debe durar alrededor de ocho a 10 semanas. Si tarda más que esto, la tarea debe subdividirse para la planeación y calendarización del proyecto.

Algunas de estas tareas se realizan paralelamente, con distintas personas que trabajan en diferentes componentes del sistema. Es necesario coordinar las tareas paralelas y organizar las actividades para que la fuerza de trabajo se desempeñe de manera óptima y no introduzca entre las tareas dependencias innecesarias. Es importante evitar una situación en la que todo el proyecto se demore debido a que una tarea crítica no está terminada.

Si un proyecto está técnicamente avanzado, las estimaciones iniciales seguramente serán optimistas aun cuando se trate de considerar todas las eventualidades. Los calendarios deben actualizarse continuamente conforme se disponga de mejor información sobre el avance. Si el proyecto a calendarizar es similar a un proyecto anterior, pueden reutilizarse las estimaciones previas.

Al evaluar calendarios hay que tomar en cuenta la posibilidad de que las cosas salgan mal. Si el proyecto es nuevo y técnicamente avanzado, partes de éste pueden resultar más difíciles y tardar más que lo previsto originalmente.

**Representación del calendario**

Los calendarios de proyecto pueden representarse simplemente en una tabla u hoja de cálculo que indique las tareas, el esfuerzo, la duración esperada y las dependencias entre las tareas. Sin embargo, este modo de representación dificulta visualizar las relaciones y dependencias entre las diferentes actividades. Por esta razón se han desarrollado representaciones gráficas alternativas de los calendarios de proyecto, que con frecuencia son más fáciles de leer y entender. Existen dos tipos de representación que se usan comúnmente:

1. **Gráficas de barras, basadas en el calendario (Gantt)**, las cuales señalan al responsable de cada actividad, el tiempo transcurrido previsto y la fecha en que se programó el inicio y el fin de la actividad.
2. **Redes de actividad**, son diagramas de red que muestran las dependencias entre las diferentes actividades que constituyen un proyecto.

Por lo general, una herramienta de planeación se usa para gestionar la información del calendario del proyecto. Dichas herramientas esperan a menudo que se introduzca la información en una tabla y luego crearán una base de datos de información del proyecto. Entonces, a partir de esta base de datos, se generan automáticamente gráficas de barras y de actividad.

Las actividades de proyecto son el elemento de planeación básico. Cada actividad cuenta con:

1. **Una duración** en días o meses calendario
2. **Una estimación del esfuerzo**, la cual refleja el número de días-hombre o meses-hombre para completar el trabajo.
3. **Un plazo** dentro del cual debe completarse la actividad
4. **Un punto final definido**. Éste representa el resultado tangible de completar la actividad.

Cuando planee un proyecto, también deberá definir los hitos; esto es, cada etapa del proyecto en la que puede realizarse una valoración del avance. Cada hito debe documentarse mediante un breve reporte que compendie el avance realizado y el trabajo efectuado. Los hitos pueden asociarse con una sola tarea o con grupos de actividades relacionadas.

Un tipo especial de hito es la producción de un entregable del proyecto. Un entregable es un producto de trabajo que se entrega al cliente. Es el resultado de una fase significativa del proyecto, como la especificación o el diseño.

Además de planear el calendario de entregas para el software, los administradores de proyecto deben asignar recursos a las tareas. Desde luego, el recurso clave son los ingenieros de software que harán el trabajo, y ellos deben asignarse a las actividades del proyecto. Las personas pueden trabajar en más de una tarea al mismo tiempo y, en ocasiones, no laborar en el proyecto. Pueden estar de vacaciones, ocupados en otros proyectos, en cursos de capacitación o inmersos en alguna otra actividad.

Si una tarea se retrasa, afectará el desarrollo de tareas posteriores que dependen de ella. No es posible iniciar éstas antes de completar la tarea retrasada. Las demoras pueden causar graves problemas con la asignación de personal, en especial cuando los individuos trabajan al mismo tiempo en varios proyectos.

**Técnicas de estimación**

Es difícil la estimación del calendario del proyecto. Probablemente haya que hacer estimaciones iniciales sobre la base de una definición de requerimientos de usuario de alto nivel. Quizá no lleguen a conocerse las personas involucradas en el proyecto y sus habilidades. Existe tanta incertidumbre que es imposible estimar con precisión los costos de desarrollo del sistema durante las primeras etapas de un proyecto.

Incluso existe una dificultad fundamental en la valoración de la precisión de diferentes enfoques a la estimación del costo y esfuerzo. Con frecuencia, las estimaciones del proyecto se autosatisfacen. La estimación se utiliza para definir el presupuesto del proyecto, y el producto se ajusta para que se cumpla la cifra del presupuesto.

Existen dos tipos de técnicas para hacer evaluaciones de esfuerzo y costo del software:

1. **Técnicas basadas en la experiencia.** La estimación de los requerimientos de esfuerzo futuro se basan en la experiencia del administrador con proyectos anteriores y el dominio de aplicación.
2. **Modelado algorítmico de costo.** Se usa un enfoque formulista para calcular el esfuerzo del proyecto con base en estimaciones de atributos del producto, así como las características del proceso.

En ambos casos es necesario usar el juicio para evaluar el esfuerzo directamente, estimar el proyecto y las características del producto. Las técnicas basadas en la experiencia dependen de la experiencia del administrador de proyectos anteriores y el esfuerzo real empleado en dichos proyectos en actividades que se relacionan con el desarrollo del software. Por lo general, se identifican los entregables que hay que producir en un proyecto y los diferentes componentes de software o sistemas a desarrollar. Esto se documenta en una hoja de cálculo, se les estima de manera individual y se calcula el esfuerzo total requerido.

La dificultad con las técnicas basadas en la experiencia es que un nuevo proyecto de software puede no tener mucho en común con proyectos anteriores. El desarrollo de software cambia muy rápidamente y con frecuencia un proyecto usará técnicas no familiares. Si usted no ha trabajado con estas técnicas, su experiencia previa puede no ser de ayuda para estimar el esfuerzo requerido, lo que dificultará producir estimaciones precisas de costo y calendario.

**Modelado algorítmico de costos**

El modelado algorítmico de costos utiliza una fórmula matemática para predecir los costos del proyecto con base en estimaciones del tamaño del proyecto, el tipo de software a desarrollar, y otros factores de equipo, proceso y producto. Un modelo algorítmico de costo puede elaborarse al analizar los costos y atributos de los proyectos completados, y encontrar la fórmula de ajuste más cercana a la experiencia real.

Los modelos algorítmicos de costo se usan principalmente para hacer estimaciones de los costos de desarrollo de software. También se pueden utilizar en la elaboración de estimaciones para inversionistas en compañías de software, estrategias alternativas para ayudar a valorar los riesgos, y decisiones informadas acerca de reutilización, re planeación o subcontratación.

Los modelos algorítmicos para estimar el esfuerzo en un proyecto de software se basan principalmente en una fórmula sencilla:

**Esfuerzo = A \* TamañoB \* M**

* A es un factor constante que depende de las prácticas locales de la organización y el tipo de software que se desarrolla.
* El tamaño puede ser una valoración del tamaño del código del software o una estimación de la funcionalidad expresada en puntos de función o de aplicación.
* El valor del exponente B se encuentra por lo general entre 1 y 1.5.
* M es un multiplicador que se integra al combinar atributos de procesos, producto y desarrollo.

El número de líneas de código fuente (SLOC) en el sistema entregado es la métrica de tamaño fundamental que se utiliza en muchos modelos algorítmicos de costo. La estimación del tamaño puede implicar estimación por analogía con otros proyectos, estimación al convertir los puntos de función al tamaño del código.

La mayoría de los modelos de estimación algorítmica tienen un componente exponencial (B en la ecuación anterior) que se relaciona con el tamaño y la complejidad del sistema. Esto refleja el hecho de que los costos no aumentan con regularidad linealmente con el tamaño del proyecto. Conforme se incrementa el tamaño y la complejidad del software, se incurre en costos adicionales debido a los costos generales de comunicación de los equipos más grandes, la administración de configuración más compleja, la integración de sistemas más difícil, etcétera. Cuanto más complejo sea el sistema, más afectarán al costo estos factores. Por lo tanto, el valor de B aumenta normalmente con el tamaño y la complejidad del sistema.

Todos los modelos algorítmicos tienen problemas similares:

1. Con frecuencia es difícil estimar el Tamaño en una etapa temprana del proyecto, cuando sólo está disponible la especificación. Las estimaciones de punto de funcion y punto de aplicación son más fáciles de producir que las estimaciones de tamaño del código, pero por lo general aún son imprecisas.
2. Las estimaciones de los factores que contribuyen a B y M son subjetivas. Las estimaciones varían de una persona a otra, dependiendo de sus antecedentes y experiencia con el tipo de sistema que se desarrolla.

La estimación precisa del tamaño del código es difícil en una etapa temprana de un proyecto, porque el tamaño del programa final depende de decisiones de diseño que pueden no haberse hecho cuando se requirió la estimación.

El lenguaje de programación usado para desarrollar el sistema afecta también el número de líneas de código a desarrollar. Un lenguaje como Java puede significar que se necesiten más líneas de código que si se usa C. Sin embargo, este código adicional permite más comprobación de tiempo de compilación, de manera que es probable que se reduzcan los costos de validación. ¿Cómo debe tomarse esto en cuenta? Más aún, se puede reutilizar una cantidad significativa de código de proyectos anteriores, así que tendrá que ajustarse la estimación del tamaño para tomar esto en cuenta.

Los modelos algorítmicos de costo son una forma sistemática de estimar el esfuerzo requerido para desarrollar un sistema. Existen muchos atributos y un margen considerable para la incertidumbre al estimar sus valores. Esta complejidad desalienta a los usuarios potenciales y, por lo tanto, la aplicación práctica del modelado algorítmico de costos se limita a un número pequeño de compañías.

Otra barrera que desalienta el uso de los modelos algorítmicos es la necesidad de calibrar sus modelos y valores de los atributos con sus datos históricos de proyecto. Sin embargo, muy pocas organizaciones recaban suficientes datos de proyectos anteriores en una forma que soporte la calibración del modelo.

Si se usa un modelo algorítmico de estimación de costos, hay que desarrollar un rango de estimaciones (peor, esperado y mejor) en lugar de una sola estimación y aplicar la fórmula de costo a todas ellas. Es más probable que las estimaciones sean precisas cuando se comprende el tipo de software que se desarrolla, se calibró el modelo de costeo usando datos locales, o cuando se predefinen el lenguaje de programación y las opciones de hardware.

**El modelo COCOMO II**

El COCOMO II es un modelo empírico que se derivó al recopilar datos a partir de un gran número de proyectos de software. Dichos datos se analizaron para descubrir qué fórmulas se ajustaban mejor con las observaciones. Dichas fórmulas vinculan el tamaño del sistema y los factores del producto, proyecto y equipo, con el esfuerzo para desarrollar el sistema. COCOMO II es un modelo de estimación bien documentado y no registrado.

COCOMO II se desarrolló a partir de los primeros modelos de estimación de costos COCOMO, que se basaron principalmente en el desarrollo de código original. El modelo COCOMO II toma en cuenta enfoques más modernos para el desarrollo de software, tales como el desarrollo rápido que usa lenguajes dinámicos, el desarrollo mediante composición de componentes y el uso de programación de base de datos. COCOMO II soporta el modelo en espiral de desarrollo, e incrusta submodelos que producen estimaciones cada vez más detalladas.

Los submodelos que se consideran parte del modelo COCOMO II son:

1. **Un modelo de composición de aplicación.** Éste modela el esfuerzo requerido para desarrollar sistemas que se crean a partir de componentes de reutilización, scripting o programación de base de datos. Las estimaciones del tamaño de software se basan en puntos de aplicación, se estiman usando una fórmula tamaño/productividad. El número de puntos de aplicación en un programa es una estimación ponderada del número de pantallas separadas que se despliegan, el número de informes que se producen, el número de módulos en lenguajes de programación imperativa y el número de líneas de lenguaje de scripting o código de programación de base de datos.
2. **Un modelo de diseño temprano.** Usado en etapas tempranas del diseño del sistema después de establecer los requerimientos. Las estimaciones se basan en puntos de función, que luego se convierten a número de líneas de código fuente. Los puntos de función son una forma independiente de lenguaje para cuantificar la funcionalidad del programa. El número total de puntos de función en un programa se calcula al medir o estimar el número de entradas y salidas externas, las interacciones de usuario, las interfaces externas y las tablas de archivos o bases de datos que usa el sistema.
3. **Un modelo de reutilización.** Este modelo se emplea para calcular el esfuerzo requerido al integrar los componentes de reutilización y/o código de programa generado automáticamente. Muchas veces se utiliza en conjunto con el modelo pos-arquitectónico.
4. **Un modelo pos-arquitectónico.** Una vez diseñada la arquitectura del sistema, puede hacerse una estimación más precisa del tamaño del software. Nuevamente, este modelo usa la fórmula estándar para estimación de costo discutida líneas arriba. Sin embargo, incluye un conjunto más extenso de 17 multiplicadores que reflejan características de capacidad personal, del producto y del proyecto.

Desde luego, en sistemas grandes pueden desarrollarse diferentes partes del sistema mediante distintas tecnologías y es posible que no se tenga que estimar todas las partes del sistema con el mismo nivel de precisión. En tales casos se puede usar el submodelo adecuado a cada parte del sistema y combinar los resultados para crear una estimación compuesta.

**El modelo de composición de aplicación**

Apoyar la estimación del esfuerzo requerido para proyectos de creación de prototipos y proyectos en que el software se desarrolla mediante la composición de los componentes existentes. Se basa en una estimación de puntos de aplicación ponderados, divididos entre una estimación estándar de productividad de puntos de aplicación. Entonces la estimación se ajusta de acuerdo con la dificultad al desarrollar cada punto de aplicación. La productividad depende de la experiencia y habilidad del desarrollador, así como de las capacidades de las herramientas de software usadas para apoyar el desarrollo.

Por lo general, la composición de aplicaciones incluye una significativa reutilización de software. Es casi seguro que ciertos puntos de aplicación en el sistema se implementen a través de componentes de reutilización. En consecuencia, habrá que ajustar la estimación para tomar en cuenta el porcentaje de reutilización previsto.

**El modelo de diseño temprano**

Este modelo puede usarse durante las primeras etapas de un proyecto, antes de que esté disponible un diseño arquitectónico detallado para el sistema. Las estimaciones de diseño temprano son más útiles para la exploración de opciones en que es necesario comparar diferentes formas de implementar los requerimientos del usuario. El modelo de diseño temprano supone que se acordaron los requerimientos del usuario y que están en marcha las etapas iniciales del proceso de diseño del sistema. La meta en esta etapa debe ser elaborar una estimación rápida y aproximada de los costos. Por lo tanto, habrá que hacer suposiciones simplificadoras.

Las estimaciones generadas en esta etapa se basan en la fórmula estándar para modelos algorítmicos, esto es:

**Esfuerzo = A \* TamañoB \* M**

**El modelo de reutilización**

La mayoría de los grandes sistemas incluyen una cantidad significativa de código que se reutilizó de proyectos de desarrollo previos. El modelo de reutilización se emplea para estimar el esfuerzo requerido al integrar código de reutilización o generado.

COCOMO II considera dos tipos de código de reutilización. El código “caja negra” es un código que puede reutilizarse sin comprender el código o hacerle cambios. Se considera que el esfuerzo de desarrollo para el código caja negra es cero. El código “caja blanca” tiene que adaptarse para integrarlo en un código nuevo u otros componentes de reutilización. Para la reutilización se requiere esfuerzo de desarrollo, pues el código tiene que entenderse y modificarse antes de que pueda trabajar correctamente en el sistema.

Muchos sistemas incluyen código generado automáticamente de modelos de sistema. Se analiza un modelo y se genera el código para implementar los objetos especificados en el modelo. El modelo de reutilización COCOMO II incluye una fórmula para estimar el esfuerzo requerido al integrar este código generado, que realiza un cálculo de esfuerzo separado. El modelo de reutilización no calcula directamente el esfuerzo a partir de una estimación del número de componentes de reutilización. En vez de ello, con base en el número de líneas de código que se reutilizan, el modelo ofrece una base para calcular el número equivalente de líneas de código nuevo. Éste se basa en el número de líneas de código de reutilización que deben cambiarse y un multiplicador que refleja la cantidad de trabajo que es necesario hacer para reutilizar los componentes.

La reutilización nunca es gratuita y se incurre en algunos costos aun si ninguna reutilización es posible. Sin embargo, los costos de reutilización disminuyen conforme aumenta la cantidad de código de reutilización.

**El nivel pos-arquitectónico**

El modelo pos-arquitectónico es el más detallado de los modelos COCOMO II. Se usa una vez que está disponible un diseño arquitectónico inicial para el sistema, de manera que se conoce la estructura del subsistema. Entonces es posible hacer estimaciones para cada parte del sistema.

El punto de partida para las estimaciones producidas en el nivel pos-arquitectónico es la misma fórmula básica usada en las estimaciones de diseño temprano.Para esta etapa del proceso, hay que hacer una estimación más precisa del tamaño del proyecto conforme se conoce cómo se dividirá el sistema en objetos o módulos. Estas estimaciones del tamaño del código se hacen mediante tres parámetros:

1. Una estimación del número total de líneas de código nuevo a desarrollar (SLOC).
2. Una estimación de los costos de reutilización, con base en un número equivalente de líneas de código fuente (ESLOC), calculadas mediante el modelo de reutilización.
3. Una estimación del número de líneas de código que es probable se modifiquen debido a cambios a los requerimientos del sistema.

Los valores de estos parámetros se suman para calcular el tamaño de código total, en KSLOC, que se emplea en la fórmula de cálculo de esfuerzo. El componente final en la estimación, el número de líneas de código modificado, refleja el hecho de que los requerimientos de software siempre cambian. Esto conduce a la reelaboración y el desarrollo de código adicional, lo que debe tomarse en cuenta. Desde luego, con frecuencia habrá incluso más incertidumbre en esta cifra que en las estimaciones de código nuevo a desarrollar.

La estimación del esfuerzo global se perfecciona usando un conjunto extenso de 17 atributos (controladores de costos) del producto, el proceso y la organización, en lugar de los siete atributos usados en el modelo de diseño temprano.

**Duración del proyecto y asignación de personal**

Además de estimar los costos globales de un proyecto y el esfuerzo que se requiere para desarrollar un sistema de software, los administradores de proyecto también deben estimar cuánto tardará el software en desarrollarse, y cuándo el personal necesitará trabajar en el proyecto. Cada vez más, las organizaciones demandan calendarios de desarrollo más cortos, de forma que sus productos puedan llegar al mercado antes que los de sus competidores.

El modelo COCOMO incluye una fórmula para estimar el tiempo calendario requerido para completar un proyecto. Sin embargo, no necesariamente son lo mismo el calendario de proyecto nominal predicho por el modelo COCOMO y el calendario requerido por el plan del proyecto. Puede haber un requerimiento para entregar el software más pronto. Si el calendario se comprime, aumenta el esfuerzo requerido para el proyecto.

La razón para esto es que agregar una persona reduce en realidad la productividad de los miembros del equipo existente y, por ende, el incremento real de esfuerzo agregado es menor que el que representa una persona. Conforme aumenta el tamaño del equipo del proyecto, los miembros del equipo pasan más tiempo comunicándose y definiendo las interfaces entre las partes del sistema desarrollado por otras personas. Por lo tanto, duplicar el número de personal (por ejemplo) no significa que la duración del proyecto se reducirá a la mitad. Si el equipo de desarrollo es grande, en ocasiones el hecho de agregar más personas al proyecto extiende, en vez de reducir, el calendario de desarrollo. Myers (1989) discute los problemas de aceleración del calendario. Él sugiere que es probable que los proyectos operen hacia problemas significativos si tratan de desarrollar software sin permitir suficiente tiempo de calendario para completar el trabajo.

No se puede simplemente estimar el número de personas requerido para un equipo de proyecto al dividir el esfuerzo total entre el tiempo contemplado en el calendario del proyecto requerido. Por lo general, al inicio del proyecto se necesita un pequeño número de personas para realizar el diseño inicial. Entonces el equipo crece hasta un nivel pico durante el desarrollo y las pruebas del sistema, y luego declina en tamaño conforme el sistema se prepara para su despliegue.

- Gestión de la Calidad

- Intereses fundamentales

1. A nivel de organización, establecer un marco de proceso y estándares de organización que

conducirán a software de mejor calidad

2. A nivel proyecto, la aplicación de procesos específicos de calidad y la verificación de que

continúen

3. A nivel proyecto, establecer un plan de calidad para un proyecto

- Aseguramiento de calidad (QA)

- Control de calidad

- Independencia del equipo de QA

- plan de calidad

1. Introducción del producto

2. Planes del producto

3. Descripciones de procesos

4. Metas de calidad

5. Riesgos y gestión del riesgo

Calidad del software

- Dificultades para determinar calidad de software

1. Es difícil escribir especificaciones de software completas y sin ambigüedades

2. Las especificaciones integran requerimientos de varias clases de participantes

3. Es imposible medir de manera directa ciertas características de calidad

- Proceso subjetivo de valoración de calidad y características no funcionales

Estándares de software

- Función de los estándares

- Importancia

1. reflejan la sabiduría que es de valor para la organización

2. proporcionan un marco para definir lo que significa el término “calidad”

3. auxilian la continuidad cuando una persona retoma el trabajo iniciado por alguien más

- Tipos de estándares

1. del producto

2. del proceso

- Alentar a la adopción de estándares

1. Involucrar a los ingenieros de software en la selección de estándares de producto

2. Revisar y modificar regularmente los estándares para reflejar las tecnologías cambiantes

3. Ofrecer herramientas de software para dar soporte a los estándares

- El marco de estándares ISO 9001

Revisiones e Inspecciones

- Definición

- El proceso de revisión

1. Actividades previas a la revisión

2. La reunión de revisión

3. Actividades posteriores a la revisión

- Inspecciones del programa

Medición y métricas del software

- Medición: definición y meta a largo plazo

- Definición de métrica

- Tipos de métricas

1. de proceso

2. de producto

- Uso de métricas

1. Para asignar un valor a los atributos de calidad del sistema

2. Para identificar los componentes del sistema cuya calidad está por debajo de un estándar

- Métricas del producto

1. Métricas dinámicas

2. Métricas estáticas

- Análisis de componentes de software

1. Elegir las mediciones a realizar

2. Seleccionar componentes a valorar

3. Medir las características de los componentes

4. Identificar mediciones anómalas

5. Analizar componentes anómalos

- Ambigüedad de mediciones

1. El software no es lo bastante bueno y no hace lo que quieren los clientes

2. El software puede ser muy bueno y se usa amplia e intensamente

**Gestión de la calidad**

La gestión de calidad del software para los sistemas de software tiene tres intereses fundamentales:

1. A nivel de organización, establecer un marco de proceso y estándares de organización que conducirán a software de mejor calidad. El equipo de gestión de calidad debe tener la responsabilidad de definir los procesos de desarrollo del software a usar, los estándares que deben aplicarse al software y la documentación relacionada, incluyendo los requerimientos, el diseño y el código del sistema.
2. A nivel del proyecto, la aplicación de procesos específicos de calidad y la verificación de que continúen dichos procesos planeados; además, se ocupa de garantizar que los resultados del proyecto estén en conformidad con los estándares aplicables a dicho proyecto.
3. A nivel del proyecto, establecer un plan de calidad para un proyecto. El plan de calidad debe establecer metas de calidad para el proyecto y definir cuáles procesos y estándares se usarán.

Los términos aseguramiento de calidad y control de calidad se utilizan ampliamente en la industria manufacturera. El aseguramiento de calidad (QA) es la definición de procesos y estándares que deben conducir a la obtención de productos de alta calidad y, en el proceso de fabricación, a la introducción de procesos de calidad. El control de calidad es la aplicación de dichos procesos de calidad para eliminar aquellos productos que no cuentan con el nivel requerido de calidad.

En la industria de software, diversas compañías y sectores industriales interpretan de maneras diferentes el aseguramiento de calidad y el control de calidad. En ocasiones, el aseguramiento de calidad representa simplemente la definición de procedimientos, procesos y estándares cuyo objetivo es asegurar el logro de calidad del software. En otros casos, el aseguramiento de calidad incluye también todas las actividades de gestión de configuración, verificación y validación aplicadas después de que un equipo de desarrollo entrega un producto.

En la mayoría de las compañías, el equipo QA es el responsable de administrar el proceso de pruebas de liberación. Esto significa que se aplican las pruebas del software antes de que éste se libere a los clientes. El equipo es responsable de comprobar que las pruebas del sistema cubran los requerimientos y de mantener los registros adecuados del proceso de pruebas.

La gestión de calidad proporciona una comprobación independiente sobre el proceso de desarrollo de software. El proceso de gestión de calidad verifica los entregables del proyecto para garantizar que sean consistentes con los estándares y las metas de la organización. El equipo QA debe ser independiente del equipo de desarrollo para que pueda tener una perspectiva objetiva del software. Esto les permite reportar la calidad del software sin estar influidos por los conflictos de desarrollo del software.

De preferencia, el equipo de gestión de calidad no debe asociarse con algún grupo de desarrollo particular; tiene la responsabilidad ante toda la organización por la administración de la calidad. El equipo debe ser independiente y reportarse ante la administración ubicada sobre el nivel del administrador del proyecto.

La planeación de calidad es el proceso de desarrollar un plan de calidad para un proyecto. El plan de calidad debe establecer las cualidades deseadas de software y describir cómo se valorarán. Por lo tanto, define lo que realmente significa software de “alta calidad” para un sistema particular. La planeación de calidad formalizada es parte integral de los procesos de desarrollo basados en un plan. No obstante, los métodos ágiles adoptan un enfoque menos formal para la gestión de calidad.

Humphrey sugiere un bosquejo de estructura para un plan de calidad. Éste incluye:

1. **Introducción del producto** Una descripción del producto, la pretensión de su mercado y las expectativas de calidad para el producto.
2. **Planes del producto** Indican las fechas de entrega críticas y las responsabilidades para el producto, junto con planes para distribución y servicio al producto.
3. **Descripciones de procesos** Describen los procesos y estándares de desarrollo y servicio que deben usarse para diseño y gestión del producto.
4. **Metas de calidad** Las metas y los planes de calidad para el producto, incluyendo una identificación y justificación de los atributos esenciales de calidad del producto.
5. **Riesgos y gestión del riesgo** Los riesgos clave que pueden afectar la calidad del producto y las acciones a tomar para enfrentar dichos riesgos.

La calidad del software puede lograrse mediante procesos establecidos basados en estándares de organización y procedimientos de calidad asociados que verifican el seguimiento de dichos estándares mediante el equipo de desarrollo de software. Su argumento es que los estándares se dirigen a la buena práctica de ingeniería de software y que seguir esta buena práctica conducirá a productos de alta calidad. No obstante, en la práctica, se considera que en la gestión de calidad hay mucho más que estándares y burocracia asociados para asegurar que éstos se sigan.

Aunque los estándares y procesos son importantes, los administradores de calidad deben enfocarse también a desarrollar una “cultura de calidad” en la que todo responsable del desarrollo del software se comprometa a lograr un alto nivel de calidad del producto. A pesar de que los estándares y procedimientos son la base de la gestión de calidad, existen aspectos intangibles a la calidad del software (elegancia, legibilidad, etcétera) que no pueden expresarse en estándares. Deben apoyar a la gente interesada en aspectos intangibles de calidad e impulsar el comportamiento profesional en todos los miembros del equipo.

La gestión de la calidad formalizada es particularmente importante para los equipos que diseñan grandes sistemas de larga duración. La documentación de la calidad es un registro de lo que cada subgrupo realizó en el proyecto. La documentación de la calidad también es un medio de comunicación durante la vida del sistema. Permite a los grupos responsables de la evolución del sistema encontrar las pruebas y comprobaciones que el equipo de desarrollo debe implementar.

Para sistemas más pequeños, la gestión de calidad es aún importante, aunque puede adoptarse un enfoque más informal. No se necesita tanto papeleo, puesto que un equipo de desarrollo pequeño puede comunicarse de manera informal. El aspecto de calidad clave para el desarrollo de sistemas pequeños es establecer una cultura de calidad y asegurarse de que todos los miembros del equipo tienen un enfoque positivo sobre la calidad del software.

**Calidad del software**

La calidad del software no es directamente comparable con la calidad en la fabricación. La idea de tolerancia no es aplicable a los sistemas digitales y es prácticamente imposible llegar a una conclusión objetiva sobre si un sistema de software cumple o no su especificación, por las siguientes razones:

1. Es difícil escribir especificaciones de software completas y sin ambigüedades. Los desarrolladores y clientes de software pueden interpretar los requerimientos de diferentes formas y tal vez sea imposible llegar a acuerdos acerca de si el software se desarrolló conforme a su especificación.
2. Las especificaciones integran requerimientos de varias clases de participantes. Dichos requerimientos son un compromiso ineludible y tal vez no incluyan los requerimientos de todos los grupos de participantes. Por lo tanto, las partes interesadas excluidas quizá perciban el sistema como uno de mala calidad, a pesar de que implementa los requerimientos acordados.
3. Es imposible medir de manera directa ciertas características de calidad y, por ende, no pueden especificarse plenamente sin ambigüedades.

Debido a estos problemas, la valoración de calidad del software es un proceso subjetivo en que el equipo de gestión de calidad tiene que usar su juicio para decidir si se logró un nivel aceptable de calidad. El equipo de gestión de calidad debe considerar si el software se ajusta o no a su propósito pretendido. Esto implica responder preguntas sobre las características del sistema. Por ejemplo:

1. ¿En el proceso de desarrollo se siguieron los estándares de programación y documentación?
2. ¿El software se verificó de manera adecuada?
3. ¿El software es suficientemente confiable para utilizarse?
4. ¿El rendimiento del software es aceptable para uso normal?
5. ¿El software es utilizable?
6. ¿El software está bien estructurado y es comprensible?

La calidad subjetiva de un sistema de software se basa principalmente en sus características no funcionales. Esto refleja la experiencia práctica del usuario: Si la funcionalidad del software no es lo que se esperaba, entonces los usuarios con frecuencia sólo le darán la vuelta a este asunto y encontrarán otras formas de hacer lo que quieren. Sin embargo, si el software no es fiable o resulta muy lento, entonces es prácticamente imposible que los usuarios logren sus metas.

Por consiguiente, la calidad del software no sólo se trata de si la funcionalidad de éste se implementó correctamente, sino también depende de los atributos no funcionales del sistema. Boehm y sus colaboradores (1978) indican que existen 15 importantes atributos de calidad de software. Dichos atributos se relacionan con la confiabilidad, usabilidad, eficiencia y mantenibilidad del software. Por lo general se considera que los atributos de confiabilidad son los atributos de calidad más importantes de un sistema. Sin embargo, también es significativo el rendimiento del software. Los usuarios rechazarán el software que sea demasiado lento.

El plan de calidad debe definir los atributos de calidad más importantes para el software que se desarrollará. Tal vez la eficiencia sea crítica y tengan que sacrificarse otros factores para que se logre esto. Si lo anterior se estableció en el plan de calidad, los ingenieros que trabajan en el desarrollo pueden cooperar para lograrlo. El plan debe incluir también una definición del proceso de valoración de la calidad. Ésta debe ser una forma acordada de valorar si cierto grado de calidad, como la mantenibilidad o robustez, está presente en el producto.

Una suposición que subyace en la gestión de la calidad del software es que la calidad del software se relaciona directamente con la calidad del proceso de desarrollo del software. Como el software se diseña en vez de manufacturarse, en el desarrollo del software es más compleja la relación entre calidad de proceso y calidad del producto. El diseño del software es un proceso creativo más que mecánico, pues es significativa la influencia de las habilidades y la experiencia individuales. No hay duda de que el proceso de desarrollo utilizado tiene una influencia importante sobre la calidad del software, y que los buenos procesos tienen más probabilidad de conducir a software de buena calidad. En consecuencia, es difícil decir cómo las características del proceso influyen en dichos atributos. Más aún, debido al papel del diseño y la creatividad en el proceso de software, la estandarización del proceso en ocasiones puede exterminar la creatividad, lo cual, lejos de elevar la calidad, conducirá a un software de calidad inferior.

**Estándares de software**

Los estándares de software tienen una función muy importante en la gestión de calidad del software. Como se indicó, un aspecto importante del aseguramiento de calidad es la selección de estándares que deben aplicarse al proceso de desarrollo o al producto. Una vez seleccionados para su uso, deben definirse procesos específicos de proyecto para monitorizar el uso de los estándares y comprobar que éstos se siguieron.

Los estándares de software son importantes por tres razones:

1. **Los estándares reflejan la sabiduría que es de valor para la organización.** Se basan en conocimiento sobre la más adecuada práctica para la compañía. Con frecuencia, este conocimiento se adquiere sólo después de gran cantidad de ensayo y error. Configurarla dentro de un estándar, ayuda a la compañía a reutilizar esta experiencia y a evitar errores del pasado.
2. **Los estándares proporcionan un marco para definir lo que significa el término “calidad”.** La calidad del software es subjetiva, y al usar estándares se establece una base para decidir si se logró un nivel de calidad requerido.
3. **Los estándares auxilian la continuidad cuando una persona retoma el trabajo iniciado por alguien más.** Los estándares aseguran que todos los ingenieros dentro de una organización adopten las mismas prácticas. En consecuencia, se reduce el esfuerzo de aprendizaje requerido al iniciarse un nuevo trabajo.

Existen dos tipos de estándares de ingeniería de software relacionados que pueden definirse y usarse en la gestión de calidad del software:

1. **Estándares del producto.** Incluyen estándares de documentos, de documentación de código y estándares de codificación.
2. **Estándares de proceso.** Establecen los procesos que deben seguirse durante el desarrollo del software. Deben especificar cómo es una buena práctica de desarrollo. Los estándares de proceso pueden incluir definiciones de especificación, procesos de diseño y validación, herramientas de soporte de proceso y una descripción de los documentos que deben escribirse durante dichos procesos.

Los estándares deben entregar valor, en la forma de calidad aumentada del producto. No hay razón para definir estándares que sean costosos en términos de tiempo y esfuerzo, pues aplicarlos sólo conduce a mejoras secundarias en la calidad. Los estándares de producto deben diseñarse de forma que puedan aplicarse y comprobarse de manera efectiva en cuanto a costos, y los estándares de proceso deben incluir la definición de procesos que comprueben que se siguieron dichos estándares.

En ocasiones, los ingenieros de software consideran los estándares como demasiado prescriptivos y realmente poco relevantes para la actividad técnica del desarrollo de software. Esto es probable sobre todo cuando los estándares de proyecto requieren documentación y registro del trabajo tediosos. Para minimizar el descontento y alentar la participación en los estándares, los administradores de calidad que establezcan los estándares deben dar los siguientes pasos:

1. **Involucrar a los ingenieros de software en la selección de estándares de producto.** Si los desarrolladores comprenden por qué se seleccionaron los estándares, tienen más probabilidad de comprometerse con éstos.
2. **Revisar y modificar regularmente los estándares para reflejar las tecnologías cambiantes.** Los estándares son costosos de desarrollar y tienden a guardarse como reliquias en un manual de estándares de una compañía. Debido a los costos y la discusión requeridos, muchas veces hay reticencia para cambiarlos. Aunque un manual de estándares es esencial, debe evolucionar para reflejar las circunstancias y la tecnología cambiantes.
3. **Ofrecer herramientas de software para dar soporte a los estándares.** Si está disponible el soporte para herramientas, se requiere muy poco esfuerzo para seguir los estándares de desarrollo de software.

Diferentes tipos de software necesitan distintos procesos de desarrollo. No hay razón para prescribir una forma particular de trabajar si es inadecuada para un proyecto o equipo de proyecto. Cada administrador de proyecto debe tener la autoridad de modificar los estándares de proceso de acuerdo con las circunstancias individuales. Es importante garantizar que dichos cambios no conduzcan a una pérdida de calidad del producto.

**El marco de estándares ISO 9001**

Existe un conjunto internacional de estándares que pueden utilizarse en el desarrollo de los sistemas de administración de calidad en todas las industrias, llamado ISO 9000. ISO 9001, el más general de dichos estándares, se aplica a organizaciones que diseñan, desarrollan y mantienen productos, incluido software.

El estándar ISO 9001 no es en sí mismo un estándar para el desarrollo de software, sino un marco para elaborar estándares de software. Establece principios de calidad total, describe en general el proceso de calidad, y explica los estándares y procedimientos organizacionales que deben determinarse. Éstos tienen que documentarse en un manual de calidad de la organización.

La revisión principal del estándar ISO 9001 reorientó el estándar hacia nueve procesos centrales. Si una organización quiere estar conforme con el estándar ISO 9001, debe documentar cómo se relacionan sus procesos con dichos procesos centrales. También deberá definir y mantener registros que demuestren que se siguieron los procesos organizacionales establecidos.

El estándar ISO 9001 no define ni prescribe los procesos de calidad específicos que deben usarse en una compañía. Para estar de conformidad con ISO 9001, una compañía debe especificar los tipos de proceso que se muestran en la figura 24.5 y tener procedimientos que demuestren que se siguen sus procesos de calidad. Esto permite flexibilidad a través de sectores industriales y diversos tamaños de compañías. Pueden definirse estándares de calidad que sean adecuados para el tipo de software a desarrollar.

Algunos clientes de software demandan que sus proveedores tengan la certificación ISO 9001. El estándar ISO 9001 se enfoca en garantizar que la organización tenga procedimientos de gestión de calidad y que siga dichos procedimientos. No hay seguridad de que las compañías con certificación ISO 9001 empleen las mejores prácticas de desarrollo de software o que sus procesos conduzcan a software de alta calidad. La certificación define la calidad como la conformidad con estándares.

Los métodos ágiles, que evitan la documentación y se enfocan en el código a desarrollar, tienen poco en común con los procesos de calidad formal que se examinan en ISO 9001. Por esta razón, las compañías que usan métodos de desarrollo ágil se preocupan pocas veces por la certificación ISO 9001.

**Revisiones e inspecciones**

Las revisiones e inspecciones son actividades QA que comprueban la calidad de los entregables del proyecto. Esto incluye examinar el software, su documentación y los registros del proceso para descubrir errores y omisiones, así como observar que se siguieron los estándares de calidad. Se usan junto con las pruebas del programa como parte del proceso general de verificación y validación del software.

Durante una revisión, un grupo de personas examinan el software y su documentación asociada en busca de problemas potenciales y la falta de conformidad con los estándares. El equipo de revisión realiza juicios informados sobre el nivel de calidad de un entregable de sistema o de proyecto. Entonces los administradores de proyecto pueden usar dichas valoraciones para tomar decisiones de planeación y asignar recursos al proceso de desarrollo.

Las revisiones de calidad se basan en documentos que se elaboraron durante el proceso de desarrollo del software. Al igual que las especificaciones, el diseño o el código del software, también pueden revisarse los modelos de proceso, planes de prueba, procedimientos de gestión de configuración, estándares de proceso y manuales de usuario. La revisión debe comprobar la coherencia e integridad de los documentos o el código objeto de prueba, y asegurarse de que se han seguido las normas de calidad.

La revisión también se utiliza para ayudar a descubrir problemas y omisiones en la documentación del software o proyecto. Las conclusiones de la revisión deben registrarse formalmente como parte del proceso de gestión de calidad. Si se descubren problemas, los comentarios de los revisores deben pasar al autor del software o a quien resulte responsable de corregir los errores u omisiones.

El propósito de las revisiones e inspecciones es mejorar la calidad del software, no de valorar el rendimiento de los miembros del equipo de desarrollo. La revisión es un proceso público de detección de errores. Es necesario que los errores cometidos por los individuos se revelen a todo el equipo de programación. Para garantizar que todos los desarrolladores participen constructivamente con el proceso de revisión, los administradores de proyecto tienen que ser sensibles a las preocupaciones individuales. Deben desarrollar una cultura de trabajo que brinde apoyo y no culpar cuando se descubran errores.

**El proceso de revisión**

El proceso de revisión se estructura por lo general en tres fases:

1. **Actividades previas a la revisión.** Se ocupan de la planeación y preparación de la revisión. La planeación de la revisión incluye establecer un equipo de revisión, organizar un tiempo, destinar un lugar para la revisión y distribuir los documentos a revisar. Durante la preparación de la revisión, el equipo puede reunirse para obtener un panorama del software a revisar.
2. **La reunión de revisión.** Durante la reunión de revisión un autor del documento o programa a revisar debe repasar el documento con el equipo de revisión.
3. **Actividades posteriores a la revisión.** Terminada una revisión, deben tratarse los conflictos y problemas surgidos durante la revisión. Esto puede implicar corregir bugs de software, refactorizar el software de modo que esté conforme con los estándares de calidad, o reescribir los documentos. Después de efectuar los cambios, la dirección de la revisión deberá comprobar que se hayan considerado todos los comentarios de la revisión.

Con frecuencia, los equipos de revisión tienen un eje de tres o cuatro personas seleccionadas como revisores principales. Un miembro debe ser un diseñador ejecutivo, quien tendrá la responsabilidad de tomar decisiones técnicas significativas. El equipo de revisión puede hacer circular el documento y pedir comentarios por escrito de un amplio número de miembros del proyecto. El administrador del proyecto necesita participar en la revisión.

Por lo general, el proceso de revisión en el desarrollo de software ágil es informal. En Scrum hay una junta de revisión después de completar cada iteración del software, en la que pueden exponerse los conflictos y problemas de calidad. En la programación extrema, la programación en grupos de dos personas garantiza que el código se examine y revise constantemente por otro miembro del equipo. Los conflictos de calidad general también se consideran en las reuniones diarias del equipo. Por lo general, los enfoques ágiles no son dirigidos por estándares, de manera que no se consideran los asuntos de cumplimiento de estándares.

**Inspecciones del programa**

Las inspecciones del programa son “revisiones de pares” en las que los miembros del equipo colaboran para encontrar bugs en el programa en desarrollo. Pueden ser parte de los procesos de verificación y validación del software. Complementan las pruebas, puesto que no requieren la ejecución del programa.

Las inspecciones del programa realizan una cuidadosa revisión del código fuente del programa. Buscan defectos y problemas, y los informan en una reunión de inspección. Los defectos pueden ser errores lógicos, anomalías en el código o ciertas características que se hayan omitido del código. Con frecuencia se usa una lista de verificación de errores comunes de programación para enfocar la búsqueda de bugs.

Los procesos ágiles pocas veces usan procesos de inspección formal o revisión de pares. En vez de ello, se apoyan en los miembros del equipo que cooperan para comprobar mutuamente el código y en lineamientos informales. Los profesionales de la programación extrema argumentan que la programación en parejas es un sustituto efectivo de la inspección, ya que, en efecto, se trata de un proceso de inspección continuo. La programación en grupos de dos conduce a un conocimiento profundo de un programa y permite encontrar bugs que a veces no se descubrirían en inspecciones formales.

**Medición y métricas del software**

La medición del software se ocupa de derivar un valor numérico o perfil para un atributo de un componente, sistema o proceso de software. Al comparar dichos valores unos con otros, y con los estándares que se aplican a través de una organización, es posible extraer conclusiones sobre la calidad del software, o valorar la efectividad de los procesos, las herramientas y los métodos de software.

La meta a largo plazo de la medición del software es usar la medición en lugar de revisiones para realizar juicios de la calidad del software. Al usar medición de software, un sistema podría valorarse preferentemente mediante un rango de métricas y, a partir de dichas mediciones, se podría inferir un valor de calidad del sistema. Si el software alcanzó un umbral de calidad requerido, entonces podría aprobarse sin revisión. Cuando es adecuado, las herramientas de medición pueden destacar también áreas del software susceptibles de mejora.

Una métrica de software es una característica de un sistema de software, documentación de sistema o proceso de desarrollo que puede medirse de manera objetiva. Los ejemplos de métricas incluyen el tamaño de un producto en líneas de código; el índice Fog, que es una medida de la legibilidad de un pasaje de texto escrito; el número de fallas reportadas en un producto de software entregado, y el número de días-hombre requerido para desarrollar un componente de sistema.

Las métricas de software pueden ser:

* **métricas de proceso:** apoyan la gestión del proceso. Se asocian por lo general con procesos de software. Ejemplos de las métricas de control son el esfuerzo promedio y el tiempo requerido para reparar los defectos reportados.
* **métricas de producto:** ayudan a predecir las características del software. Se asocian con el software en sí. Ejemplos: la complejidad ciclomática de un módulo, la longitud promedio de los identificadores de un programa, el número de atributos y operaciones asociados con las clases de objetos en un diseño.

Las métricas pueden influir en la toma de decisiones administrativas. Los administradores usan mediciones de proceso para decidir si deben hacerse cambios al proceso, y las métricas de producto ayudan a estimar el esfuerzo requerido para hacer cambios al software.

Existen dos formas en que pueden usarse las mediciones de un sistema de software:

1. **Para asignar un valor a los atributos de calidad del sistema.** Al medir las características de los componentes del sistema, como su complejidad ciclomática, y luego agregar dichas mediciones, es posible valorar los atributos de calidad del sistema, tales como la mantenibilidad.
2. **Para identificar los componentes del sistema cuya calidad está por debajo de un estándar.** Las mediciones pueden identificar componentes individuales con características que se desvían de la norma.

Lamentablemente, es difícil hacer mediciones directas de muchos de los atributos de calidad del software. Los atributos de calidad son atributos externos que se refieren a cómo los desarrolladores y usuarios experimentan el software. Se ven afectados por factores subjetivos, como la experiencia y educación del usuario, y no pueden medirse de manera objetiva. Para hacer un juicio sobre estos atributos, hay que medir algunos atributos internos del software (como tamaño, complejidad, etcétera) y suponer que éstos se relacionan con las características de calidad por las que uno se interesa.

La medición y las métricas de software son la base de la ingeniería de software empírica. Ésta es un área de investigación en la que se han usado experimentos respecto a los sistemas de software, y la recolección de datos referente a proyectos reales para formar y validar hipótesis sobre métodos y técnicas de ingeniería de software.

**Métricas del producto**

Las métricas del producto son métricas de predicción usadas para medir los atributos internos de un sistema de software. Los ejemplos de las métricas de productos incluyen el tamaño del sistema, la medida en líneas de código o el número de métodos asociados con cada clase de objeto. Por desgracia, las características del software que pueden medirse fácilmente, como el tamaño y la complejidad ciclomática, no tienen una relación clara y consistente con los atributos de calidad como comprensibilidad y mantenibilidad. Las relaciones varían dependiendo de los procesos de desarrollo, la tecnología empleada y el tipo de sistema a diseñar.

Las métricas del producto se dividen en dos clases:

1. **Métricas dinámicas.** Se recopilan mediante mediciones hechas de un programa en ejecución. Dichas métricas pueden recopilarse durante las pruebas del sistema o después de que el sistema está en uso. Un ejemplo es el número de reportes de bugs o el tiempo necesario para completar un cálculo.
2. **Métricas estáticas.** Se recopilan mediante mediciones hechas de representaciones del sistema, como el diseño, el programa o la documentación. Ejemplos de mediciones estáticas son el tamaño del código y la longitud promedio de los identificadores que se usaron.

Estos tipos de métrica se relacionan con diferentes atributos de calidad. Las métricas dinámicas ayudan a valorar la eficiencia y fiabilidad de un programa. Las métricas estáticas ayudan a valorar la complejidad, comprensibilidad y mantenibilidad de un sistema de software o de los componentes del sistema.

Por lo general, existe una relación clara entre métricas dinámicas y características de calidad del software. Es muy sencillo medir el tiempo de ejecución requerido para funciones particulares y valorar el tiempo requerido con la finalidad de iniciar un sistema. Éstos se relacionan directamente con la eficiencia del sistema. De igual modo, el número de fallas del sistema y el tipo de fallas pueden registrarse y relacionarse directamente con la fiabilidad del software.

Las métricas estáticas tienen una relación indirecta con los atributos de calidad. El tamaño del programa y la complejidad del control parecen ser los factores de predicción más fiables de la comprensibilidad, la complejidad del sistema y la mantenibilidad en base a experimentos realizados

**Análisis de componentes de software**

Se ilustra un proceso de medición que puede ser parte de un proceso de valoración de calidad del software. Cada componente del sistema puede analizarse por separado mediante un rango de métricas. Los valores de dichas métricas pueden compararse entonces para diferentes componentes. Las mediciones anómalas pueden implicar que existen problemas con la calidad de dichos componentes.

Las etapas clave en este proceso de medición de componentes son:

1. **Elegir las mediciones a realizar.** Deben formularse las preguntas que la medición busca responder, y definir las mediciones requeridas para responder a tales preguntas.
2. **Seleccionar componentes a valorar** Probablemente no se necesite estimar valores métricos para todos los componentes en un sistema de software. Tal vez desee enfocarse en los componentes centrales del sistema que están casi en uso constante.
3. **Medir las características de los componentes.** Se miden los componentes seleccionados y se calculan los valores de métrica asociados.
4. **Identificar mediciones anómalas.** Después de hacer las mediciones de componentes, se comparan entonces unas con otras y con mediciones anteriores que se hayan registrado en una base de datos de mediciones. Hay que observar los valores inusualmente altos o bajos para cada métrica, pues éstos sugieren que podría haber problemas con el componente que muestra dichos valores.
5. **Analizar componentes anómalos.** Cuando identifique los componentes con valores anómalos para sus métricas seleccionadas, debe examinarlos para decidir si dichos valores de métrica anómalos significan que la calidad del componente se encuentra o no comprometida. Un valor de métrica anómalo para la complejidad (al parecer) no necesariamente significa un componente de mala calidad. Podría haber alguna otra razón para el valor alto, por lo que no necesariamente significa que haya problemas con la calidad del componente.

Siempre es conveniente mantener datos recopilados como un recurso organizacional, así como registros históricos de todos los proyectos aun cuando no se hayan usado durante un proyecto particular. Una vez establecida una base suficientemente grande de datos de medición, será posible hacer comparaciones de calidad de software a través de proyectos, además de validar las relaciones entre atributos de componentes internos y características de calidad.

**Ambigüedad de mediciones**

Cuando reúna datos cuantitativos relativos al software y los procesos de software, deberá analizar dichos datos para entender su significado. Es fácil malinterpretar los datos y hacer inferencias incorrectas. No basta con observar los datos por sí mismos, sino que también hay que considerar el contexto donde se recaban los datos.

Para comprender por qué puede ocurrir ambigüedad, hay que conocer las razones por las que los usuarios pueden hacer peticiones de cambio:

1. **El software no es lo bastante bueno y no hace lo que quieren los clientes.** Por lo tanto, solicitan cambios para obtener la funcionalidad que ellos requieren.
2. **El software puede ser muy bueno y se usa amplia e intensamente.** Las peticiones de cambio pueden generarse porque existen muchos usuarios de software que piensan creativamente en nuevas ideas que podrían hacer con el software.

Los cambios al proceso pueden conducir a muchos nuevos clientes satisfechos que deseen participar en el proceso de desarrollo del producto. Por lo tanto, generan más peticiones de cambio. Los cambios al proceso de manejar las peticiones de cambio contribuyen a este aumento. Si la compañía tiene mayor capacidad de respuesta con los clientes, ellos generarán más peticiones de cambio porque saben que éstas se tomarán con seriedad. Creen que sus sugerencias se incorporarán quizás en versiones posteriores del software. O bien, el número de peticiones de cambio puede aumentar porque los sitios de prueba beta no eran los típicos del mayor uso del programa.

Para analizar los datos de petición de cambio, no basta con conocer el número de peticiones de cambio, sino que se necesita conocer quién hizo la petición, cómo usa el software y por qué hizo la petición. También se requiere información sobre los factores externos, como modificaciones al procedimiento de petición de cambio o cambios al mercado que puedan tener un efecto. Con esta información, es posible averiguar si los cambios al proceso fueron efectivos para aumentar la calidad del producto.

Esto ilustra las dificultades de entender los efectos de los cambios, y el enfoque “científico” a este problema es reducir el número de factores que tiendan a afectar las mediciones hechas. Sin embargo, los procesos y productos que se miden no están aislados de su entorno. El ambiente empresarial cambia constantemente y es imposible evitar los cambios a la práctica laboral sólo porque pueden hacerse comparaciones de datos inválidos. Como tales, los datos cuantitativos sobre las actividades humanas no siempre deben tomarse en serio. Las razones por las que cambia un valor medido con frecuencia son ambiguas. Dichas razones deben investigarse a profundidad antes de extraer conclusiones de cualquier medición que se haya realizado.