

GUIA DE PREGUNTAS

Material "GESTION DE PROYECTOS.
Ingeniería del Software (Capítulo 3) de Roger S. Pressman (5^{ta} Edición)"

1. Enuncie en que se centra la gestión eficaz de un proyecto de software.
2. Caracterice el modelo de madurez de la capacidad gestión de personal.
3. Caracterice producto software (objetivos y ámbito).
4. Caracterice proceso software.
5. Caracterice proyecto software.
6. Enuncie y describa categorías de participantes que componen el proceso del software.
7. Defina el modelo de gestión de proyectos MOI.
8. Enuncie opciones que pueden aplicarse a los recursos humanos de un proyecto que requiere "n" personas trabajando durante "k" años.
9. Enuncie los organigramas de equipo genéricos propuestos por Mantei.
10. Fundamente la utilidad de cada organigrama.
11. Enuncie los factores de un proyecto propuestos por Mantei que deberían considerarse cuando se planifica el organigrama de equipos de ingeniería del software.
12. Enuncie los paradigmas de organización para equipos de ingeniería del software propuestos por Constantine.
13. Enuncie premisas para conseguir un equipo de alto rendimiento.
14. Enuncie factores que fomentan un entorno de equipo tóxico potencial según Jackman.
15. Grafique valor y empleo de técnicas de coordinación y comunicación.
16. Enuncie las categorías de técnicas de coordinación de proyectos propuestas por Kraul y Streeter.
17. Enuncie las cuestiones que definen el ámbito del software.
18. Enuncie modelos de proceso.
19. Enuncie actividades estructurales de la organización de producción de software.
20. Defina señales que indican que un proyecto de sistemas de información está en peligro según Reel.
21. Enuncie las partes que Reel sugiere como aproximación a proyectos de software.
22. Enuncie las preguntas propuestas por Bohem que conducen a la definición de las características clave del proyecto y el plan del proyecto resultante.
23. Enuncie las preguntas que el Concilio Airlie ha desarrollado para tener una "Visión Rápida" sobre si un proyecto ha implementado prácticas críticas.

CONCEPTOS SOBRE GESTIÓN DE PROYECTOS

EN el prefacio de su libro sobre gestión de proyectos de software, Meiler Page-Jones [PAG85] dice una frase que pueden corroborar muchos asesores de ingeniería del software:

He visitado docenas de empresas, buenas y malas, y he observado a muchos administradores de proceso de datos, también buenos y malos. Frecuentemente, he visto con horror cómo estos administradores **luchaban** inútilmente contra proyectos de pesadilla, sufriendo por fechas límite imposibles de cumplir, o que entregaban sistemas que decepcionaban a **sus** usuarios y que devoraban ingentes cantidades de horas de mantenimiento.

Lo que describe Page-Jones son síntomas que resultan de una serie de problemas técnicos y de gestión. Sin embargo, si se hiciera una autopsia de cada proyecto, es muy probable que se encontrara un denominador común constante: una débil gestión.

En este capítulo y en los seis que siguen consideraremos los conceptos clave que llevan a una gestión efectiva de proyectos de software. Este capítulo trata conceptos y principios básicos sobre gestión de proyectos de software. El Capítulo 4 presenta las métricas del proyecto y del proceso, la base para una toma de decisiones de gestión efectivas. Las técnicas que se emplean para estimar los costes y requisitos de recursos y poder establecer un plan efectivo del proyecto se estudian en el Capítulo 5. Las actividades de gestión que llevan a una correcta supervisión, reducción y gestión del riesgo se presentan en el Capítulo 6. El Capítulo 7 estudia las actividades necesarias para definir las tareas de un proyecto y establecer una programación del proyecto realista. Finalmente, los Capítulos 8 y 9 consideran técnicas para asegurar la calidad a medida que se dirige un proyecto y el control de los cambios a lo largo de la vida de una aplicación.

VISTAZO RÁPIDO

¿Qué es? Aunque muchos de nosotros (en nuestros momentos más oscuros) tomamos la visión de Dilbert sobre la «gestión», falta una actividad muy necesaria cuando se construyen productos y sistemas basados en computadoras. La gestión de proyectos implica la planificación, supervisión y control del personal, del proceso y de los eventos que ocurren mientras evoluciona el software desde la fase preliminar a la implementación operacional.

¿Quién lo hace? Todos «gestionamos» de algún modo, pero el ámbito de las actividades de gestión varía en función de la persona que las realiza. Un ingeniero de software gestiona sus actividades del día a día, planificando, supervisando, y controlando las tareas técnicas. Los gestores del proyecto planifican, supervisan y controlan el trabajo de un equipo de ingenieros de software. Los gestores expertos coordi-

nan la relación entre el negocio y los profesionales del software.

¿Por qué es importante? La construcción de software de computadora es una empresa compleja, particularmente si participa mucha gente, trabajando durante un periodo de tiempo relativamente largo. Esta es la razón por la cual los proyectos de software necesitan ser gestionados.

¿Cuáles son los pasos? Comprender las cuatro «P's» —personal, producto, proceso y proyecto—. El personal debe estar organizado para desarrollar el trabajo del software con efectividad. La comunicación con el cliente debe ocurrir para que se comprendan el alcance del producto y los requisitos. Debe seleccionarse el proceso adecuado para el personal, y el producto. El proyecto debe planificarse estimando el esfuerzo y el tiempo para cumplir las tareas; definiendo los productos del trabajo, estable-

ciendo puntos de control de calidad y estableciendo mecanismos para controlar y supervisar el trabajo definido en la planificación.

¿Cuál es el producto obtenido? Un plan de proyecto se realiza al comienzo de las actividades de gestión. El plan define el proceso y las tareas a realizar el personal que realizará el trabajo y los mecanismos para evaluar los riesgos, controlar el cambio y evaluar la calidad.

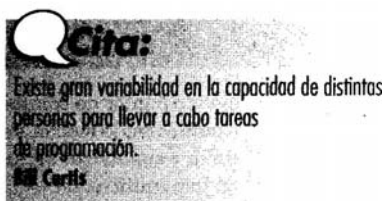
¿Cómo puedo estar seguro de que lo he hecho correctamente? Nunca estás completamente seguro de que el plan de proyecto es correcto hasta que no has entregado un producto de alta calidad dentro del tiempo y del presupuesto. Sin embargo, un gestor de proyectos hace lo correcto cuando estimula al personal para trabajar juntos como un equipo efectivo, centrandó su atención en las necesidades del cliente y en la calidad del producto.

3.1 EL ESPECTRO DE LA GESTIÓN

La gestión eficaz de un proyecto de software se centra en las cuatro P's: *personal, producto, proceso y proyecto*. El orden no es arbitrario. El gestor que se olvida de que el trabajo de ingeniería del software es un esfuerzo humano intenso nunca tendrá éxito en la gestión de proyectos. Un gestor que no fomenta una minuciosa comunicación con el cliente al principio de la evolución del proyecto se arriesga a construir una elegante solución para un problema equivocado. El administrador que presta poca atención al proceso corre el riesgo de arrojar métodos técnicos y herramientas eficaces al vacío. El gestor que emprende un proyecto sin un plan sólido arriesga el éxito del producto.

3.1.1. Personal

La necesidad de contar con personal para el desarrollo del software altamente preparado y motivado se viene discutiendo desde los años 60 (por ejemplo, [COU80, WIT94, DEM98]). De hecho, el «factor humano» es tan importante que el Instituto de Ingeniería del Software ha desarrollado un *Modelo de madurez de la capacidad de gestión de personal* (MMCGP) «para aumentar la preparación de organizaciones del software para llevar a cabo las cada vez más complicadas aplicaciones ayudando a atraer, aumentar, motivar, desplegar y retener el talento necesario para mejorar su capacidad de desarrollo de software» [CUR94].



El modelo de madurez de gestión de personal define las siguientes áreas clave prácticas para el personal que desarrolla software: reclutamiento, selección, gestión de rendimiento, entrenamiento, retribución, desarrollo de la carrera, diseño de la organización y del trabajo y desarrollo cultural y de espíritu de equipo.

El MMCGP es compañero del modelo de madurez de la capacidad software (Capítulo 2), que guía a las organizaciones en la creación de un proceso de software maduro. Más adelante en este capítulo se consideran aspectos asociados a la gestión de personal y estructuras para proyectos de software.

3.1.2. Producto

Antes de poder planificar un proyecto, se deberían establecer los objetivos y el ámbito del producto¹, se deberían considerar soluciones alternativas e identificar las dificultades técnicas y de gestión. Sin esta información, es imposible definir unas estimaciones razonables (y exactas) del coste; una valoración efectiva del riesgo, una subdivisión realista de las tareas del proyecto o una planificación del proyecto asequible que proporcione una indicación fiable del progreso.

Referencia cruzada

En el Capítulo 1 se trata una taxonomía de las áreas de aplicación que producen los «productos» de software.

El desarrollador de software y el cliente deben reunirse para definir los objetivos del producto y su ámbito. En muchos casos, esta actividad empieza como parte del proceso de ingeniería del sistema o del negocio (Capítulo 10) y continúa como el primer paso en el análisis de los requisitos del software (Capítulo 11). Los objetivos identifican las metas generales del proyecto sin considerar cómo se conseguirán (desde el punto de vista del cliente).

El ámbito identifica los datos primarios, funciones y comportamientos que caracterizan al producto, y, más importante, intenta *abordar* estas características de una manera cuantitativa.

Una vez que se han entendido los objetivos y el ámbito del producto, se consideran soluciones alternativas.

3.1.3. Proceso

Un proceso de software (Capítulo 2) proporciona la estructura desde la que se puede establecer un detallado plan para el desarrollo del software. Un pequeño número de actividades estructurales se puede aplicar a todos los proyectos de software, sin tener en cuenta su tamaño o complejidad. Diferentes conjuntos de tareas —tareas, hitos, productos del trabajo y puntos de garantía de calidad— permiten a las actividades estructurales adaptarse a las características del proyecto de software y a los requisitos del equipo del proyecto. Finalmente, las actividades protectoras —tales como garantía de calidad del software, gestión de la configuración del software y medición— cubren el modelo de proceso. Las actividades protectoras son independientes de las estructurales y tienen lugar a lo largo del proceso.

¹ En este contexto, el término «producto» es usado para abarcar cualquier software que será construido a petición de otros. Esto incluye no sólo productos de software, sino también sistemas basados en computadora, software empujado y software de resolución de problemas (por ejemplo, programas para la resolución de problemas científicos/de ingeniería).

PUNTO CLAVE

Las actividades estructurales se componen de tareas, hitos, productos de trabajo y puntos de garantía de calidad.

3.1.4. Proyecto

Dirigimos los proyectos de software planificados y controlados por una razón principal —es la Única manera conocida de gestionar la complejidad—. Y todavía seguimos esforzándonos. En 1998, los datos de la industria del software indicaron que el 26 por 100 de proyectos de software fallaron completamente y que el 46

por 100 experimentaron un desbordamiento en la planificación y en el coste [REE99]. Aunque la proporción de éxito para los proyectos de software ha mejorado un poco, nuestra proporción de fracaso de proyecto permanece más alto del que debería ser².

Para evitar el fracaso del proyecto, un gestor de proyectos de software y los ingenieros de software que construyeron el producto deben eludir un conjunto de señales de peligro comunes; comprender los factores del éxito críticos que conducen a la gestión correcta del proyecto y desarrollar un enfoque de sentido común para planificar, supervisar y controlar el proyecto. Cada uno de estos aspectos se trata en la Sección 3.5 y en los capítulos siguientes.

3.2 PERSONAL

En un estudio publicado por el IEEE [CUR88] se les preguntó a los vicepresidentes ingenieros de tres grandes compañías tecnológicas sobre el factor más importante que contribuye al éxito de un proyecto de software. Respondieron de la siguiente manera:

VP 1: Supongo que si tuviera que elegir lo más importante de nuestro entorno de trabajo, diría que no son las herramientas que empleamos, es la gente.

VP 2: El ingrediente más importante que contribuyó al éxito de este proyecto fue tener gente lista ... pocas cosas más importan en mi opinión ... Lo más importante que se puede hacer por un proyecto es seleccionar el personal ... El éxito de la organización de desarrollo del software está muy, muy asociado con la habilidad de reclutar buenos profesionales.

VP 3: La única regla que tengo en cuanto a la gestión es asegurarme de que tengo buenos profesionales —genterealmente buena—, de que preparo buena gente y de que proporcione el entorno en el que los buenos profesionales puedan producir.

Cita:
Las compañías que gestionan sensiblemente su inversión en personal a la larga prosperarán.
Tom DeMarco y Tim Lister

Ciertamente, éste es un testimonio convincente sobre la importancia del personal en el proceso de ingeniería del software. Y, sin embargo, todos nosotros, desde los veteranos vicepresidentes al más modesto profesional

del software, damos este aspecto por descontado. Los gestores argumentan (como el grupo anterior) que el personal es algo primario, pero los hechos desmienten a veces sus palabras.

En esta sección examinamos los participantes que colaboran en el proceso del software y la manera en que se organizan para realizar una ingeniería del Software eficaz.

3.2.1. Los participantes

El proceso del software (y todos los proyectos de software) lo componen participantes que pueden clasificarse en una de estas cinco categorías:


1. *Gestores superiores*, que definen los aspectos de negocios que a menudo tienen una significativa influencia en el proyecto.
2. *Gestores (técnicos) del proyecto*, que deben planificar, motivar, organizar y controlar a los profesionales que realizan el trabajo de software.
3. *Profesionales*, que proporcionan las capacidades técnicas necesarias para la ingeniería de un producto o aplicación.
4. *Clientes*, que especifican los requisitos para la ingeniería del software y otros elementos que tienen menor influencia en el resultado.
5. *Usuarios finales*, que interaccionan con el software una vez que se ha entregado para la producción.

Para ser eficaz, el equipo del proyecto debe organizarse de manera que maximice las habilidades y capacidades de cada persona. Y este es el trabajo del jefe del equipo.

² Dadas estas estadísticas, es razonable preguntarse cómo el impacto de las computadoras continúa creciendo exponencialmente y la industria del software continúa anunciando el crecimiento de ventas al doble. Parte de la respuesta, pienso, es que un importante número de estos proyectos «fallidos» están mal concebidos desde el primer momento. Los clientes pierden el interés rápidamente (puesto que lo que ellos pidieron realmente no era tan importante como ellos habían pensado), y los proyectos son cancelados.

3.2.2. Los jefes de equipo

La gestión de un proyecto es una actividad intensamente humana, y por esta razón, los profesionales competentes del software a menudo no son buenos jefes de equipo. Simplemente no tienen la mezcla adecuada de capacidades del personal. Y sin embargo, como dice Edgemon: «Desafortunadamente y con demasiada frecuencia, hay individuos que terminan en la gestión de proyectos y se convierten en gestores de proyecto accidentales» [EDG95].


 **¿En qué nos fijamos cuando seleccionamos a alguien para conducir un proyecto de software?**

En un excelente libro sobre gestión técnica, Jerry Weinberg [WEI86] sugiere el modelo de gestión MOI:

Motivación. La habilidad para motivar (con un «tira y afloja») al personal técnico para que produzca conforme a sus mejores capacidades.

Organización. La habilidad para amoldar procesos existentes (o inventar unos nuevos) que permita al concepto inicial transformarse en un producto final.

Ideas o innovación. La habilidad para motivar al personal para crear y sentirse creativos incluso cuando deban de trabajar dentro de los límites establecidos para un producto o aplicación de software particular.

 **Cita:**
En equipos sencillos un líder es aquel que sabe a donde quiere ir, se levanta y va.
John Erskine

Weinberg sugiere que el éxito de los gestores de proyecto se basa en aplicar un estilo de gestión en la resolución de problemas. Es decir, un gestor de proyectos de software debería concentrarse en entender el problema que hay que resolver, gestionando el flujo de ideas y, al mismo tiempo, haciendo saber a todos los miembros del equipo (mediante palabras y, mucho más importante, con hechos) que la calidad es importante y que no debe verse comprometida.

Otro punto de vista [EDG95] de las características que definen a un gestor de proyectos eficiente resalta cuatro apartados clave:

Resolución del problema. Un gestor eficiente de un proyecto de software puede diagnosticar los aspectos técnicos y de organización más relevantes, estructurar una solución sistemáticamente o motivar apropiadamente a otros profesionales para que desarrollen la solución, aplicar las lecciones aprendidas de anteriores proyectos a las nuevas situaciones, mantenerse lo suficientemente flexible para cambiar la gestión si los intentos iniciales de resolver el problema no dan resultado.

Dotes de gestión. Un buen gestor de proyectos debe tomar las riendas. Debe tener confianza para asumir el control cuando sea necesario y la garantía para permitir que los buenos técnicos sigan sus instintos.

Incentivos por logros. Para optimizar la productividad de un equipo de proyecto, un gestor debe recompensar la iniciativa y los logros, y demostrar a través de sus propias acciones que no se penalizará si se corren riesgos controlados.

Influencia y construcción de espíritu de equipo. Un gestor de proyecto eficiente debe ser capaz de «leer» a la gente; debe ser capaz de entender señales verbales y no verbales y reaccionar ante las necesidades de las personas que mandan esas señales. El gestor debe mantener el control en situaciones de gran estrés.



Un experto de software puede no tener temperamento o deseo de ser jefe de equipo. No fuerce al experto para ser uno de ellos.

3.2.3. El equipo de software

Existen casi tantas estructuras de organización de personal para el desarrollo de software como organizaciones que se dedican a ello. Para bien o para mal, el organigrama no puede cambiarse fácilmente. Las consecuencias prácticas y políticas de un cambio de organización no están dentro del alcance de las responsabilidades del gestor de un proyecto de software. Sin embargo, la organización del personal directamente involucrado en un nuevo proyecto de software está dentro del ámbito del gestor del proyecto.

Las siguientes opciones pueden aplicarse a los recursos humanos de un proyecto que requiere n personas trabajando durante k años:

1. n individuos son asignados a m diferentes tareas funcionales, tiene lugar relativamente poco trabajo conjunto; la coordinación es responsabilidad del gestor del software que puede que tenga otros seis proyectos de los que preocuparse.



Cita:
No todo grupo es un equipo, y no todo equipo es eficiente.
Glenn Parker

2. n individuos son asignados a m diferentes tareas funcionales ($m < n$) de manera que se establecen «equipos informales»; se puede nombrar un líder al efecto; la coordinación entre los equipos es responsabilidad de un gestor del software.
3. n individuos se organizan en t equipos; a cada equipo se le asignan una o más tareas funcionales; cada equipo tiene una estructura específica que se define para todos los equipos que trabajan en el proyecto; la coordinación es controlada por el equipo y por el gestor del proyecto de software.

Aunque es posible encontrar argumentos en pro y en contra para cada uno de los enfoques anteriores, existe

una gran evidencia que indica que una organización de equipo formal (opción 3) es la más productiva.

La «mejor» estructura de equipo depende del estilo de gestión de una organización, el número de personas que compondrá el equipo, sus niveles de preparación y la dificultad general del problema. Mantei [MAN81] sugiere tres organigramas de equipo genéricos:

Descentralizado democrático (DD). Este equipo de ingeniería del software no tiene un jefe permanente. Más bien, «se nombran coordinadores de tareas a corto plazo y se sustituyen por otros para diferentes tareas». Las decisiones sobre problemas y los enfoques se hacen por consenso del grupo. La comunicación entre los miembros del equipo es horizontal.



¿Cómo debería estar organizado un equipo de software?

Descentralizado controlado (DC). Este equipo de ingeniería del software tiene un jefe definido que coordina tareas específicas y jefes secundarios que tienen responsabilidades sobre subtarear. La resolución de problemas sigue siendo una actividad del grupo, pero la implementación de soluciones se reparte entre subgrupos por el jefe de equipo. La comunicación entre subgrupos e individuos es horizontal. También hay comunicación vertical a lo largo de la jerarquía de control.

Centralizado controlado (CC). El jefe del equipo se encarga de la resolución de problemas a alto nivel y la coordinación interna del equipo. La comunicación entre el jefe y los miembros del equipo es vertical.

Mantei [MAN81] describe siete factores de un proyecto que deberían considerarse cuando se planifica el organigrama de equipos de ingeniería del software:

- la dificultad del problema que hay que resolver
- el tamaño del programa(s) resultante(s) en líneas de código o puntos de función (Capítulo 4)
- el tiempo que el equipo estará junto (tiempo de vida del equipo)
- el grado en que el problema puede ser modularizado



¿Qué factores deberíamos considerar cuando estructuramos un equipo de software?

- la calidad requerida y fiabilidad del sistema que se va a construir
- la rigidez de la fecha de entrega
- el grado de sociabilidad (comunicación) requerido para el proyecto

Debido a que una estructura centralizada realiza las tareas más rápidamente, es la más adecuada para mane-

jar problemas sencillos. Los equipos descentralizados generan más y mejores soluciones que los individuales. Por tanto, estos equipos tienen más probabilidades de éxito en la resolución de problemas complejos. Puesto que el equipo DC es centralizado para la resolución de problemas, tanto el organigrama de equipo DC como el de CC pueden aplicarse con éxito para problemas sencillos. La estructura DD es la mejor para problemas difíciles.

Como el rendimiento de un equipo es inversamente proporcional a la cantidad de comunicación que se debe entablar, los proyectos muy grandes son mejor dirigidos por equipos con estructura CC o DC, donde se pueden formar fácilmente subgrupos.

El tiempo que los miembros del equipo vayan a «vivir juntos» afecta a la moral del equipo. Se ha descubierto que los organigramas de equipo tipo DD producen una moral más alta y más satisfacción por el trabajo y son, por tanto, buenos para equipos que permanecerán juntos durante mucho tiempo.

El organigrama de equipo DD se aplica mejor a problemas con modularidad relativamente baja, debido a la gran cantidad de comunicación que se necesita. Los organigramas CC o DC funcionan bien cuando es posible una modularidad alta (y la gente puede hacer cada uno lo suyo).



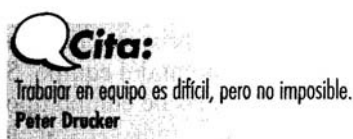
Frecuentemente es mejor tener pocos equipos pequeños, bien centrados que un gran equipo solamente.

Los equipos CC y DC producen menos defectos que los equipos DD, pero estos datos tienen mucho que ver con las actividades específicas de garantía de calidad que aplica el equipo. Los equipos descentralizados requieren generalmente más tiempo para completar un proyecto que un organigrama centralizado y al mismo tiempo son mejores cuando se precisa una gran cantidad de comunicación.

Constantine [CON93] sugiere cuatro «paradigmas de organización» para equipos de ingeniería del software:

1. **Un paradigma cerrado** estructura a un equipo con una jerarquía tradicional de autoridad (similar al equipo CC). Estos equipos trabajan bien cuando producen software similar a otros anteriores, pero probablemente sean menos innovadores cuando trabajen dentro de un paradigma cerrado.
2. **El paradigma aleatorio** estructura al equipo libremente y depende de la iniciativa individual de los miembros del equipo. Cuando se requiere innovación o avances tecnológicos, los equipos de paradigma aleatorio son excelentes. Pero estos equipos pueden chocar cuando se requiere un «rendimiento ordenado».

3. *El paradigma abierto* intenta estructurar a un equipo de manera que consiga algunos de los controles asociados con el paradigma cerrado, pero también mucha de la innovación que tiene lugar cuando se utiliza el paradigma aleatorio. El trabajo se desarrolla en colaboración, con mucha comunicación y toma de decisiones consensuadas y con el sello de los equipos de paradigma abierto. Los organigramas de equipo de paradigma abierto son adecuados para la resolución de problemas complejos, pero pueden no tener un rendimiento tan eficiente como otros equipos.



4. *El paradigma sincronizado* se basa en la compartimentación natural de un problema y organiza los miembros del equipo para trabajar en partes del problema con poca comunicación activa entre ellos.

Constantine [CON93] propone una variación en el equipo descentralizado democrático defendiendo a los equipos con independencia creativa cuyo enfoque de trabajo podría ser mejor llamado «anarquía innovadora». Aunque se haya apelado al enfoque de libre espíritu para el desarrollo del software, el objetivo principal de una organización de Ingeniería del Software debe ser «convertir el caos en un equipo de alto rendimiento» [HYM93]. Para conseguir un equipo de alto rendimiento.

- Los miembros del equipo deben confiar unos en otros.
- La distribución de habilidades debe adecuarse al problema.
- Para mantener la unión del equipo, los inconformistas tienen que ser excluidos del mismo

Cualquiera que sea la organización del equipo, el objetivo para todos los gestores de proyecto es colaborar a crear un equipo que presente cohesión. En su libro, *Peopfeware*, DeMarco y Lister [DEM98] estudian este aspecto:

Referencia cruzada

El papel del bibliotecario existe sin tener en cuenta la estructura del equipo. Para más detalles véase el Capítulo 9.

Tendemos a usar la palabra equipo demasiado libremente en el mundo de los negocios, denominando «equipo» a cualquier grupo de gente asignado para trabajar junta. Pero muchos de estos grupos no parecen equipos. No tie-

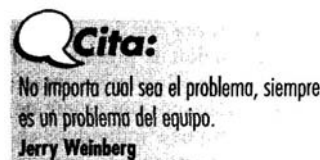
nen una definición común de éxito o un espíritu de equipo identificable. Lo que falta es un fenómeno que denominamos *cuajar*.

Un equipo cuajado es un grupo de gente tejido tan fuertemente que el todo es mayor que la suma de las partes...

Una vez que el equipo empieza a cuajar, la probabilidad de éxito empieza a subir. El equipo puede convertirse en imparable, un monstruo de éxito ... No necesitan ser dirigidos de una manera tradicional y no necesitan que se les motive. Están en su gran momento.

DeMarco y Lister mantienen que los miembros de equipos cuajados son significativamente más productivos y están más motivados que la media. Comparten una meta común, una cultura común y, en muchos casos, un «sentimiento elitista» que les hace únicos.

Pero no todos los equipos cuajan. De hecho, muchos equipos sufren lo que Jackman llama «toxicidad de equipo» [JAC98] define cinco factores que «fomentan un entorno de equipo tóxico potencial»:



1. una atmósfera de trabajo frenética en la que los miembros del equipo gastan energía y se descentran de los objetivos del trabajo a desarrollar;
2. alta frustración causada por factores tecnológicos, del negocio, o personales que provocan fricción entre los miembros del equipo;
3. «procedimientos coordinados pobremente o fragmentados» o una definición pobre o impropriamente elegida del modelo de procesos que se convierte en un obstáculo a saltar;
4. definición confusa de los papeles a desempeñar produciendo una falta de responsabilidad y la acusación correspondiente, y
5. «continua y repetida exposición al fallo» que conduce a una pérdida de confianza y a una caída de la moral.

Jackman sugiere varios antitóxicos que tratan los problemas comunes destacados anteriormente.

Para evitar un entorno de trabajo frenético, el gestor de proyectos debería estar seguro de que el equipo tiene acceso a toda la información requerida para hacer el trabajo y que los objetivos y metas principales, una vez definidos, no deberían modificarse a menos que fuese absolutamente necesario. Además, las malas noticias no deberían guardarse en secreto, sino entregarse al equipo tan pronto como fuese posible (mientras haya tiempo para reaccionar de un modo racional y controlado).



los equipos formados son lo ideal, pero no es fácil conseguirlos. Como mínimo, esté seguro de evitar un «entorno tóxico».

Aunque la frustración tiene muchas causas, los desarrolladores de software a menudo la sienten cuando pierden la autoridad para controlar la situación. Un equipo de software puede evitar la frustración si recibe tanta responsabilidad para la toma de decisiones como sea posible. Cuanto más control se le da al equipo para tomar decisiones técnicas y del proceso, menos frustración sentirán los miembros del equipo.

Una elección inapropiada del proceso del software (p.ej., tareas innecesarias o pesadas, pobre elección de los productos del trabajo) puede ser evitada de dos formas: (1) estando seguros de que las características del software a construir se ajustan al rigor del proceso elegido, y (2) permitiendo al equipo seleccionar el proceso (con el reconocimiento completo de que, una vez elegido, el equipo tiene la responsabilidad de entregar un producto de alta calidad).

El gestor de proyectos de software, trabajando junto con el equipo, debería refinar claramente los roles y las responsabilidades antes del comienzo del proyecto. El equipo debería establecer sus propios mecanismos para la responsabilidad (las revisiones técnicas formales³ son una forma para realizar esto) y definir una serie de enfoques correctivos cuando un miembro del equipo falla en el desarrollo.

Todo equipo de software experimenta pequeños fallos. La clave para eliminar una atmósfera de fallos será establecer técnicas basadas en el equipo para retroalimentar y solucionar el problema. Además, cualquier fallo de un miembro del equipo debe ser considerado como un fallo del equipo. Esto lleva a un acercamiento del equipo a la acción correctiva, en lugar de culpar y desconfiar, que ocurre con rapidez en equipos tóxicos.



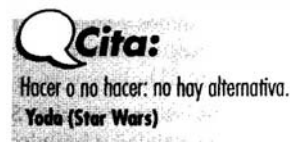
¿Cómo debemos evitar «toxinas» que con frecuencia infectan un equipo de software?

Además de las cinco toxinas descritas por Jackman, un equipo de software a menudo se enfrenta con los rasgos humanos diferentes de sus miembros. Algunos miembros del equipo son extrovertidos, otros son introvertidos. Algunas personas recogen información intuitivamente, separando amplios conceptos de hechos dispares. Otros procesan la información linealmente, reuniendo y organizando detalles minuciosos de los datos proporcionados. Algunos miembros del equipo toman las decisiones apropiadas solamente cuando se

presenta un argumento lógico, de un modo ordenado. Otros son intuitivos, pudiendo tomar una decisión basándose en sus «sensaciones». Algunos desarrolladores prefieren una planificación detallada compuesta por tareas organizadas que les permita lograr el cierre para algún elemento de un proyecto. Otros prefieren un entorno más espontáneo donde aspectos abiertos son correctos. Algunos trabajan duro para tener las cosas hechas mucho antes de la fecha de un hito, de ese modo eliminan la presión cuando se aproximan a las fechas, mientras que otros están apurados por las prisas para hacer la entrega en el Último minuto. Un estudio detallado de la psicología de estos rasgos y de las formas en las que un jefe de equipo cualificado puede ayudar a la gente con rasgos opuestos para trabajar juntos está fuera del ámbito de este libro⁴. Sin embargo, es importante destacar que el reconocimiento de las diferencias humanas es el primer paso hacia la creación de equipos que cuajan.

3.2.4. Aspectos sobre la coordinación y la comunicación

Hay muchos motivos por los que los proyectos de software pueden tener problemas. La *escala* (tamaño) de muchos esfuerzos de desarrollo es grande, conduciendo a complejidades, confusión y dificultades significativas para coordinar a los miembros del equipo. La *incertidumbre* es corriente, dando como resultado un continuo flujo de cambios que impactan al equipo del proyecto. La *interoperatividad* se ha convertido en una característica clave de muchos sistemas. El software nuevo debe comunicarse con el anterior y ajustarse a restricciones predefinidas impuestas por el sistema o el producto.



Estas características del software moderno —escala, incertidumbre e interoperatividad— son aspectos de la vida. Para enfrentarse a ellos eficazmente, un equipo de ingeniería del software debe establecer métodos efectivos para coordinar a la gente que realiza el trabajo. Para lograr esto se deben establecer mecanismos de comunicación formales e informales entre los miembros del equipo y entre múltiples equipos. La comunicación formal se lleva a cabo «por escrito, con reuniones organizadas y otros canales de comunicación relativamente no interactivos e impersonales» [KRA95]. La comunicación informal es más personal. Los miembros de un equipo de software comparten ideas de por sí, piden ayuda a medida que surgen los problemas e interactúan los unos con los otros diariamente.

³ Las revisiones técnicas formales se tratan con detalle en el Capítulo 8.

⁴ Se puede encontrar una excelente introducción a estos temas relacionados con los equipos de proyectos de software en [FER98]

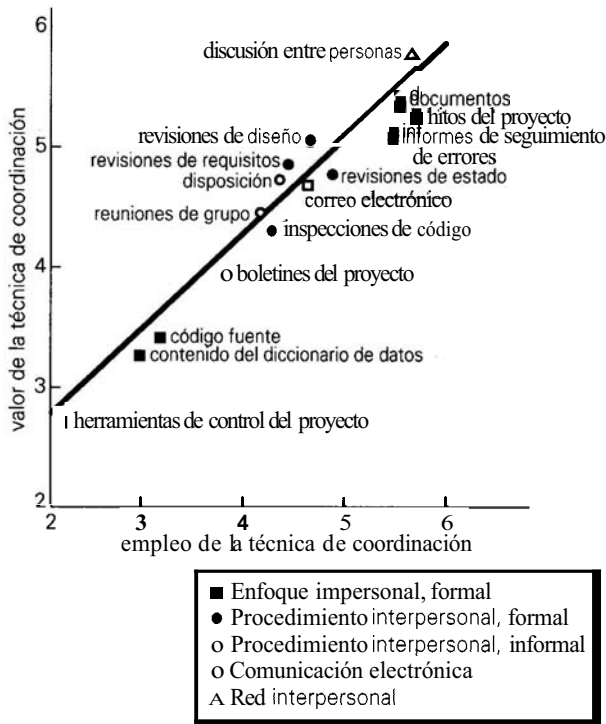


FIGURA 3.1. Valor y empleo de técnicas de coordinación y comunicación.

Kraul y Streeter [KRA95] examinan una colección de técnicas de coordinación de proyectos que se categorizan de la siguiente manera:

Formal, enfoque impersonal. Incluyen documentos de ingeniería del software y entregas (incluyendo el código fuente), memorandos técnicos, hitos del proyecto, planificaciones del programa y herramientas de control del proyecto (Capítulo 7), peticiones de cambios y documentación relativa (Capítulo 9), informes de seguimiento de errores e información almacenada (Capítulo 31).

Formal, procedimientos interpersonales. Se centra en las actividades de garantía de calidad (Capítulo 8) aplicada

a productos de ingeniería del software. Estos incluyen reuniones de revisión de estado e inspecciones de diseño y de código.

¿Cómo coordinar las acciones de los miembros del equipo?

Informal, procedimientos interpersonales. Incluyen reuniones de grupo para la divulgación de información y resolución de problemas así como «definición de requisitos y del personal de desarrollo».

Comunicación electrónica. Comprende correo electrónico, boletines de noticias electrónicos y, por extensión, sistemas de videoconferencia.

Red interpersonal. Discusiones informales con los miembros del equipo y con personas que no están en el proyecto pero que pueden tener experiencia o una profunda visión que puede ayudar a los miembros del equipo.

Para valorar la eficacia de estas técnicas para la coordinación de proyectos, Kraul y Streeter estudiaron 65 proyectos de software con cientos de personas implicadas. La Figura 3.1 (adaptada de [KRA95]) expresa el valor y empleo de las técnicas de coordinación apuntadas anteriormente. En la figura, el valor percibido (clasificado en una escala de siete puntos) de varias técnicas de comunicación y coordinación es contrastado con su frecuencia de empleo en un proyecto. Las técnicas situadas por encima de la línea de regresión fueron «juzgadas como relativamente valiosas, dado la cantidad de veces que se emplearon» [KRA95]. Las técnicas situadas por debajo de la línea se consideraron de menor valor. Es interesante resaltar que las redes interpersonales fueron catalogadas como las técnicas de mayor valor de coordinación y de comunicación. Es también importante hacer notar que los primeros mecanismos de garantía de calidad del software (requisitos y revisiones de diseño) parecieron tener más valor que evaluaciones posteriores de código fuente (inspecciones de código).

3.3 PRODUCTO

El gestor de un proyecto de software se enfrenta a un dilema al inicio de un proyecto de ingeniería del software. Se requieren estimaciones cuantitativas y un plan organizado, pero no se dispone de información sólida. Un análisis detallado de los requisitos del software proporcionaría la información necesaria para las estimaciones, pero el análisis a menudo lleva semanas o meses. **Aún peor, los requisitos pueden ser fluidos, cambiando regularmente a medida que progresa el proyecto. Y, aún así, se necesita un plan «¡ya!».**

Por tanto, debemos examinar el producto y el problema a resolver justo al inicio del proyecto. Por lo menos se debe establecer el ámbito del producto y delimitarlo.

3.3.1. Ámbito del software

La primera actividad de gestión de un proyecto de software es determinar *el ámbito del software*. El ámbito se define respondiendo a las siguientes cuestiones:



Si no puede delimitar una característica de software que intento construir, considere la característica como un riesgo principal del proyecto.

Contexto. ¿Cómo encaja el software a construir en un sistema, producto o contexto de negocios mayor y qué limitaciones se imponen como resultado del contexto?

Objetivos de información. ¿Qué objetos de datos visibles al cliente (Capítulo 11) se obtienen del software? ¿Qué objetos de datos son requeridos de entrada?

Función y rendimiento. ¿Qué función realiza el software para transformar la información de entrada en una salida? ¿Hay características de rendimiento especiales que abordar?

El ámbito de un proyecto de software debe ser unívoco y entendible a niveles de gestión y técnico. Los enunciados del ámbito del software deben estar delimitados.

Es decir, los datos cuantitativos (por ejemplo: número de usuarios simultáneos, tamaño de la lista de correo, máximo tiempo de respuesta permitido) se establecen explícitamente; se anotan las limitaciones (por ejemplo: el coste del producto limita el tamaño de la memoria) y se describen los factores de reducción de riesgos (por ejemplo: los algoritmos deseados se entienden muy bien si están disponibles en C++).

3.3.2. Descomposición del problema

La descomposición del problema, denominado a veces *particionado* o *elaboración del problema*, es una actividad que se asienta en el núcleo del análisis de requisitos del software (Capítulo 11). Durante la actividad de exposición del ámbito no se intenta descomponer el problema totalmente. Más bien, la descomposición se aplica en dos áreas principales: (1) la funcionalidad que debe entregarse y (2) el proceso que se empleará para entregarlo.



Para desarrollar un plan de proyecto razonable, tiene que descomponer funcionalmente el problema a resolver

Los seres humanos tienden a aplicar la estrategia de divide y vencerás cuando se enfrentan a problemas complejos. Dicho de manera sencilla, un problema complejo se parte en problemas más pequeños que resultan más manejables. Ésta es la estrategia que se aplica al inicio de la planificación del proyecto.

Las funciones del software, descritas en la exposición del ámbito, se evalúan y refinan para proporcionar más detalle antes del comienzo de la estimación (Capítulo 5). Puesto que ambos, el coste y las estimaciones de la planificación temporal, están orientados funcionalmente, un pequeño grado de descomposición suele ser útil.

Por ejemplo, considere un proyecto que construirá un nuevo procesador de textos. Entre las características peculiares del producto están: la introducción de información a través de la voz así como del teclado; características extremadamente sofisticadas de «edición automática de copia»; capacidad de diseño de página; indexación automática y tabla de contenido, y otras. El gestor del proyecto debe establecer primero la exposición del ámbito que delimita estas características (así como otras funciones más normales, como edición, administración de archivos, generación de documentos). Por ejemplo, ¿requerirá la introducción de información mediante voz «entrenamiento» por parte del usuario? ¿Qué capacidades específicas proporcionará la característica de editar copias? ¿Exactamente cómo será de sofisticado la capacidad de diseño de página?

Referencia cruzada

En el Capítulo 12 se presenta una técnica útil para descomponer el problema, llamada «análisis gramatical».

A medida que evoluciona la exposición del ámbito, un primer nivel de partición ocurre de forma natural. El equipo del proyecto sabe que el departamento de marketing ha hablado con clientes potenciales y ha averiguado que las siguientes funciones deberían ser parte de la edición automática de copia: (1) comprobación ortográfica; (2) comprobación gramatical; (3) comprobación de referencias para documentos grandes (p. ej.: ¿se puede encontrar una referencia a una entrada bibliográfica en la lista de entradas de la bibliografía?), y (4) validación de referencias de sección y capítulo para documentos grandes. Cada una de estas características representa una subfunción para implementar en software. Cada una puede ser aún más refinada si la descomposición hace más fácil la planificación.

3.4 PROCESO

Las fases genéricas que caracterizan el proceso de software definición, desarrollo y soporte — son aplicables a todo software. El problema es seleccionar el modelo de proceso apropiado para la ingeniería del software que debe aplicar el equipo del proyecto. En el Capítulo 2 se estudió una gran gama de paradigmas de ingeniería del software:

- el modelo secuencial lineal
- el modelo de prototipo
- el modelo DRA

- el modelo incremental
- el modelo en espiral
 - el modelo en espiral WINWIN
- el modelo de desarrollo basado (ensamblaje) en componentes
 - el modelo de desarrollo concurrente
- el modelo de métodos formales
 - el modelo de técnicas de cuarta generación



Una vez elegido el modelo de proceso, acompáñelo con el mínimo conjunto de tareas de trabajo y productos que desembocaron en un producto de alta calidad -evite la capacidad de destrucción del proceso.

El gestor del proyecto debe decidir qué modelo de proceso es el más adecuado para (1) los clientes que han solicitado el producto y la gente que realizará el trabajo; (2) las características del producto en sí, y (3) el entorno del proyecto en el que trabaja el equipo de software. Cuando se selecciona un modelo de proceso, el equipo define entonces un plan de proyecto preliminar basado en un conjunto de actividades estructurales. Una vez establecido el plan preliminar, empieza la descomposición del proceso. Es decir, se debe crear un plan completo reflejando las tareas requeridas a las personas para cubrir las actividades estructurales. Exploramos estas actividades brevemente en las secciones que siguen y presentamos una visión **más** detallada en el Capítulo 7.

3.4.1. Maduración del producto y del proceso

La planificación de un proyecto empieza con la maduración del producto y del proceso. Todas las funciones que se deben tratar dentro de un proceso de ingeniería por el equipo de software deben pasar por el conjunto de actividades estructurales que se han definido para una organización de software. Asuma que la organización ha adoptado el siguiente conjunto de actividades estructurales (Capítulo 2):

- *Comunicación con el cliente* — tareas requeridas para establecer la obtención de requisitos eficiente entre el desarrollador y el cliente.
- *Planificación* — tareas requeridas para definir los recursos, la planificación temporal del proyecto y cualquier información relativa a él.



Recuerde..... las actividades estructurales se aplican en todos los proyectos- no hay excepciones.

- *Análisis del riesgo* — tareas requeridas para valorar los riesgos técnicos y de gestión.
- *Ingeniería* — tareas requeridas para construir una o más representaciones de la aplicación.
- *Construcción y entrega* — tareas requeridas para construir, probar, instalar y proporcionar asistencia al usuario (por ejemplo: documentación y formación).
- *Evaluación del cliente* — tareas requeridas para obtener información de la opinión del cliente basadas en la evaluación de las representaciones de software creadas durante la fase de ingeniería e implementadas durante la fase de instalación.

Los miembros del equipo que trabajan en cada función aplicarán todas las actividades estructurales. En esencia, se crea una matriz similar a la que se muestra en la Figura 3.2. Cada función principal del problema (la figura contiene las funciones para el software procesador de textos comentado anteriormente) se lista en la columna de la izquierda. Las actividades estructurales se listan en la fila

ACTIVIDADES ESTRUCTURALES DE PROCESO COMUNES																				
	comunicación con el cliente				planificación				análisis de riesgo				ingeniería							
Tareas de Ingeniería del Software																				
Funciones del producto																				
Introducción de texto																				
Edición y formateo																				
Edición automática de copia																				
Capacidad de diseño de página																				
Indexación automática y TOC																				
Administración de archivos																				
Producción de documentos																				

FIGURA 3.2. Maduración del problema y del proceso.

de arriba. Las tareas de trabajo de ingeniería del software (para cada actividad estructural) se introducirían en la fila siguiente⁵. El trabajo del gestor del proyecto (y de otros miembros del equipo) es estimar los requisitos de recursos para cada celda de la matriz, poner fechas de inicio y finalización para las tareas asociadas con cada celda y los productos a fabricar como consecuencia de cada celda. Estas actividades se consideran en los Capítulos 5 y 7.

CLAVE

La descomposición del producto y del proceso se produce simultáneamente con la evolución del plan de proyecto.

3.4.2. Descomposición del proceso

Un equipo de software debería tener un grado significativo de flexibilidad en la elección del paradigma de ingeniería del software que resulte mejor para el proyecto y de las tareas de ingeniería del software que conforman el modelo de proceso una vez elegido. Un proyecto relativamente pequeño similar a otros que se hayan hecho anteriormente se debería realizar con el enfoque secuencial lineal. Si hay límites de tiempo muy severos y el problema se puede compartimentar mucho, el modelo apropiado es el DRA (en inglés RAD). Si la fecha límite está tan próxima que no va a ser posible entregar toda la funcionalidad, una estrategia incremental puede ser lo mejor. Similarmente, proyectos con otras características (p. ej.: requisitos inciertos, nuevas tecnologías, clientes difíciles, potencialidad de reutilización) llevarán a la selección de otros modelos de proceso⁶.



Aplique siempre la ECP (Estructura Común de Proceso), sin tener en cuenta el tamaño, criticidad o tipo del proyecto. Las tareas pueden variar pero la ECP no.

Una vez que se ha elegido el modelo de proceso, la estructura común de proceso (ECP) se adapta a él. En todos los casos, el ECP estudiado anteriormente en este capítulo —comunicación con el cliente, planificación, análisis de riesgo, ingeniería, construcción, entrega y evaluación del cliente— puede adaptarse al paradigma. Funcionará para modelos lineales, para modelos iterativos e incrementales, para modelos de evolución e incluso para modelos concurrentes o de ensamblaje de componentes. El ECP es invariable y sirve como base para todo el trabajo de software realizado por una organización.

Pero las tareas de trabajo reales sí varían. La descomposición del proceso comienza cuando el gestor del proyecto pregunta: «¿Cómo vamos a realizar esta actividad

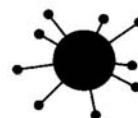
ECP?». Por ejemplo, un pequeño proyecto, relativamente simple, requiere las siguientes tareas para la actividad de *comunicación con el cliente*:

1. Desarrollar una lista de aspectos que se han de clarificar.
2. Reunirse con el cliente para resolver los aspectos que se han de clarificar.
3. Desarrollar conjuntamente una exposición del ámbito del proyecto.
4. Revisar el alcance del proyecto con todos los implicados.
5. Modificar el alcance del proyecto cuando se requiera.

Estos acontecimientos pueden ocurrir en un periodo de menos de 48 horas. Representan una descomposición del problema apropiado para proyectos pequeños relativamente sencillos.

Ahora, consideremos un proyecto más complejo que tenga un ámbito más amplio y un mayor impacto comercial. Un proyecto como ése podría requerir las siguientes tareas para la actividad de comunicación con el cliente:

1. Revisar la petición del cliente.
2. Planificar y programar una reunión formal con el cliente.
3. Realizar una investigación para definir soluciones propuestas y enfoques existentes.
4. Preparar un «documento de trabajo» y una agenda para la reunión formal.
5. Realizar la reunión.
6. Desarrollar conjuntamente mini-especificaciones que reflejen la información, función y características de comportamiento del software.



Modelo de proceso adaptable.

7. Revisar todas las mini-especificaciones para comprobar su corrección, su consistencia, la ausencia de ambigüedades.
8. Ensamblar las mini-especificaciones en un documento de alcance del proyecto.
9. Revisar ese documento general con todo lo que pueda afectar.
10. Modificar el documento de alcance del proyecto cuando se requiera.

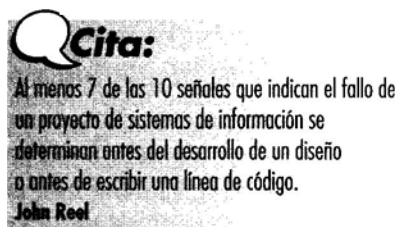
Ambos proyectos realizan la actividad estructural que denominamos *comunicación con el cliente*, pero el equipo del primer proyecto lleva a cabo la mitad de tareas de ingeniería del software que el segundo.

⁵ Se hace notar que las tareas se deben adaptar a las necesidades específicas de un proyecto. Las actividades estructurales siempre permanecen igual, pero las tareas se seleccionarán basándose en unos criterios de adaptación. Este tema es discutido más adelante en el Capítulo 7 y en la página Web SEPA 5/e.

⁶ Recuerde que las características del proyecto tienen también una fuerte influencia en la estructura del equipo que va a realizar el trabajo. Vea la Sección 3.2.3.

3.5 PROYECTO

Para gestionar un proyecto de software con éxito, debemos comprender qué puede ir mal (para evitar esos problemas) y cómo hacerlo bien. En un excelente documento sobre proyectos de software, John Reel [REE99] define diez señales que indican que un proyecto de sistemas de información está en peligro:



1. La gente del software no comprende las necesidades de los clientes.
2. El ámbito del producto está definido pobremente.
3. Los cambios están mal realizados.
4. La tecnología elegida cambia.
5. Las necesidades del negocio cambian [o están mal definidas].
6. Las fechas de entrega no son realistas.
7. Los usuarios se resisten.
8. Se pierden los patrocinadores [o nunca se obtuvieron adecuadamente].
9. El equipo del proyecto carece del personal con las habilidades apropiadas.
10. Los gestores [y los desarrolladores] evitan buenas prácticas y sabias lecciones.

Los profesionales veteranos de la industria hacen referencia frecuentemente a la regla 90-90 cuando estudian proyectos de software particularmente difíciles: el primer 90 por 100 de un sistema absorbe el 90 por 100 del tiempo y del esfuerzo asignado. El último 10 por 100 se lleva el otro 90 por 100 del esfuerzo y del tiempo asignado [ZAH94]. Los factores que conducen a la regla 90-90 están contenidos en las diez señales destacadas en la lista anterior.

¡Suficiente negatividad! ¿Cómo actúa un gestor para evitar los problemas señalados anteriormente? Reel [REE99] sugiere una aproximación de sentido común a los proyectos de software dividida en cinco partes:

Empezar con el pie derecho. Esto se realiza trabajando duro (muy duro) para comprender el problema a solucionar y estableciendo entonces objetivos y expectativas realistas para cualquiera que

vaya a estar involucrado en el proyecto. Se refuerza construyendo el equipo adecuado (Sección 3.2.3) y dando al equipo la autonomía, autoridad y tecnología necesaria para realizar el trabajo.

Mantenerse. Muchos proyectos no realizan un buen comienzo y entonces se desintegran lentamente. Para mantenerse, el gestor del proyecto debe proporcionar incentivos para conseguir una rotación del personal mínima, el equipo debería destacar la calidad en todas las tareas que desarrolle y los gestores veteranos deberían hacer todo lo posible por permanecer fuera de la forma de trabajo del equipo⁷.

Seguimiento del Progreso. Para un proyecto de software, el progreso se sigue mientras se realizan los productos del trabajo (por ejemplo, especificaciones, código fuente, conjuntos de casos de prueba) y se aprueban (utilizando revisiones técnicas formales) como parte de una actividad de garantía de calidad. Además, el proceso del software y las medidas del proyecto (capítulo 4) pueden ser reunidas y utilizadas para evaluar el progreso frente a promedios desarrollados por la organización de desarrollo de software.

Tomar Decisiones Inteligentes. En esencia, las decisiones del gestor del proyecto y del equipo de software deberían «seguir siendo sencillas». Siempre que sea posible, utilice software del mismo comercial o componentes de software existentes; evite personalizar interfaces cuando estén disponibles aproximaciones estándar; identifique y elimine entonces riesgos obvios; asigne más tiempo del que pensaba necesitar para tareas arriesgadas o complejas (necesitará cada minuto).



Se puede encontrar un gran conjunto de recursos que pueden ayudar tanto a gestores de proyecto experimentados como a principiantes en:
www.pmi.org, www.4pm.com y
www.projectmanagement.com

Realizar un Análisis «Postmortem» (después de finalizar el proyecto). Establecer un mecanismo consistente para extraer sabias lecciones de cada proyecto. Evaluar la planificación real y la prevista, reunir y analizar métricas del proyecto de software y realimentar con datos de los miembros del equipo y de los clientes, y guardar los datos obtenidos en formato escrito.


⁷ Esta frase implica la reducción al mínimo de la burocracia, y la eliminación tanto de reuniones extrañas como de la adherencia dogmática a las reglas del proceso y del proyecto. El equipo debena estar capacitado para realizar esto.

1. EL PRINCIPIO W⁵HH

En un excelente documento sobre proyectos y proceso del software, Barry Boehm [BOE96] afirma: «... se necesita un principio de organización que haga una simplificación con el fin de proporcionar planes [de proyectos] sencillos para proyectos pequeños». Boehm sugiere un enfoque que trate los objetivos, hitos y planificación, responsabilidades, enfoque técnico y de gestión, y **los** recursos requeridos del proyecto. Boehm lo llama el principio «WWW⁵HH», después de una serie de preguntas (7 cuestiones) que conducen a la definición de las características clave del proyecto y el plan del proyecto resultante:

¿Por qué se desarrolla el sistema? La respuesta a esta pregunta permite a todas las partes evaluar la validez de las razones del negocio para el trabajo del software. Dicho de otra forma, ¿justifica el propósito del negocio el gasto en personal, tiempo, y dinero?

¿Qué se realizará cuándo? La respuesta a estas preguntas ayuda al equipo a establecer la planificación del proyecto identificando las tareas clave del proyecto y los hitos requeridos por el cliente.

 ¿Qué preguntas necesitan ser respondidas para desarrollar un Plan de Proyecto?

¿Quién es el responsable de una función? Antes en este capítulo, señalamos que el papel y la res-

ponsabilidad de cada miembro del equipo de software debe estar definida. La respuesta a la pregunta ayuda a cumplir esto.

¿Dónde están situados organizacionalmente? No todos los roles y responsabilidades residen en el equipo de software. El cliente, los usuarios, y otros directivos también tienen responsabilidades.



Plan de Proyecto de Software

¿Cómo estará realizado el trabajo desde el punto de vista técnico y de gestión? Una vez establecido el ámbito del producto, se debe definir una estrategia técnica y de gestión para el proyecto.

¿Qué cantidad de cada recurso se necesita? La respuesta a esta pregunta se deriva de las estimaciones realizadas (Capítulo 5) basadas en respuestas a las preguntas anteriores.

El principio W⁵HH de Boehm es aplicable sin tener en cuenta el tamaño o la complejidad del proyecto de software. Las preguntas señaladas proporcionan un perfil de planificación al gestor del proyecto y al equipo de software.

1.7 PRÁCTICAS CRÍTICAS

El Concilio⁸ Airlie ha desarrollado una lista de «prácticas críticas de software para la gestión basada en el rendimiento». Estas prácticas son «utilizadas de un modo consistente por, y consideradas críticas por, organizaciones y proyectos de software de mucho éxito cuyo rendimiento “final” es más consistente que los promedios de la industria» [AIR99]. En un esfuerzo por permitir a una organización de software determinar si un proyecto específico ha implementado prácticas críticas, el Concilio Airlie ha desarrollado un conjunto de preguntas de «Visión Rápida» [AIR99] para un proyecto⁹:

Gestión formal del riesgo. ¿Cuáles son los diez riesgos principales para este proyecto? Para cada

uno de **los** riesgos ¿cuál es la oportunidad de que el riesgo se convierta en un problema y cuál es el impacto si lo hace?

Coste empírico y estimación de la planificación. ¿Cuál es el tamaño actual estimado de la aplicación de software (sin incluir el software del sistema) que será entregada en la operación? ¿Cómo se obtuvo?



Visión rápida del Proyecto Airlie

⁸ El Concilio Airlie es un equipo de expertos en ingeniería del software promocionado por el Departamento de Defensa U.S. ayudando en el desarrollo de directrices para prácticas importantes en la gestión de proyectos de software y en la ingeniería del Software.

⁹ Solo se destacan aquí aquellas practicas críticas relacionadas con la «integridad del proyecto»). Otras practicas mejores se trataran en capítulos posteriores.

Gestión de proyectos basada en métricas. ¿Dispone de un programa de métricas para dar una primera indicación de los problemas del desarrollo? Si es así, ¿cuál es la volatilidad de los requisitos actualmente?

Seguimiento del valor ganado. ¿Informa mensualmente de las métricas del valor ganado...? Si es así, ¿están calculadas estas métricas desde una red de actividades de tareas para el esfuerzo total a la próxima entrega?

Seguimiento de defectos frente a objetivos de calidad. ¿Realiza el seguimiento e informa periódicamente del número de defectos encontrados en cada prueba de inspección [revisión técnica formal] y ejecución des-

de el principio del programa y del número de defectos que se corrigen y se producen en la actualidad?

Gestión del programa del personal. ¿Cuál es la media de rotación de la plantilla en los tres últimos meses por cada uno de los distribuidores/desarrolladores involucrados en el desarrollo del software para este sistema?

Si un equipo de proyectos de software no puede responder a estas preguntas, o las responde inadecuadamente, se debe realizar una revisión completa de las prácticas del proyecto. Cada una de las prácticas críticas señaladas anteriormente se tratan con detalle a lo largo de la Parte II de este libro.

RESUMEN

La gestión de proyectos de software es una actividad protectora dentro de la ingeniería del software. Empieza antes de iniciar cualquier actividad técnica y continúa a lo largo de la definición, del desarrollo y del mantenimiento del software.

Hay cuatro «P's» que tienen una influencia sustancial en la gestión de proyectos de software —personal, producto, proceso y proyecto—. El personal debe organizarse en equipos eficaces, motivados para hacer un software de alta calidad y coordinados para alcanzar una comunicación efectiva. Los requisitos del producto deben comunicarse desde el cliente al desarrollador, dividirse (descomponerse) en las partes que lo constituyen y distribuirse para que trabaje el equipo de software. El proceso debe adaptarse al personal y al problema. Se selecciona una estructura común de proceso, se aplica un paradigma apropiado de ingeniería del software y se elige un conjunto de tareas para com-

pletar el trabajo. Finalmente, el proyecto debe organizarse de una manera que permita al equipo de software tener éxito.

El elemento fundamental en todos los proyectos de software es el personal. Los ingenieros del software pueden organizarse en diferentes organigramas de equipo que van desde las jerarquías de control tradicionales a los equipos de «paradigma abierto». Se pueden aplicar varias técnicas de coordinación y comunicación para apoyar el trabajo del equipo. En general, las revisiones formales y las comunicaciones informales persona a persona son las más valiosas para los profesionales.

La actividad de gestión del proyecto comprende medición y métricas, estimación, análisis de riesgos, planificación del programa, seguimiento y control. Cada uno de estos aspectos se trata en los siguientes capítulos.

REFERENCIAS

- [AIR99] Airlie Council, «Performance Based Management: The Program Manager's Guide Based on the 16-Point Plan and Related Metrics», Draft Report, 8 de Marzo, 1999.
- [BAK72] Baker, F.T., «Chief Programmer Team Management of Production Programming», *IBM Systems Journal*, vol. 11, n.º 1, 1972, pp. 56-73.
- [BOE96] Boehm, B., «Anchoring the Software Process», *IEEE Software*, vol. 13, n.º 4, Julio de 1996, pp. 73-82.
- [CON93] Constantine, L., «Work Organization: Paradigms for Project Management and Organization», *CACM*, vol. 36, n.º 10, Octubre de 1993, pp. 34-43.
- [COU80] Cougar, J., y R. Zawacky, *Managing and Motivating Computer Personnel*, Wiley, 1980.
- [CUR88] Curtis, B., *et al.*, «A Field Study of the Software Design Process for Large Systems», *IEEE Trans. Software Engineering*, vol. 31, n.º 11, Noviembre de 1988, pp. 1268-1287.
- [CUR94] Curtis, B., *et al.*, *People Management Capability Maturity Model*, Software Engineering Institute, Pittsburgh, PA, 1994.
- [DEM98] DeMarco, T., y T. Lister, *Peopleware*, 2.ª edición, Dorset House, 1998.
- [EDG95] Edgemon, J., «Right Stuff How to Recognize It When Selecting a Project Manager», *Application Development Trends*, vol. 2, n.º 5, Mayo de 1995, pp. 37-42.
- [FER98] Ferdinandi, P. L., «Facilitating Communication», *IEEE Software*, Septiembre de 1998, pp. 92-96.
- [JAC98] Jackman, M., «Homeopathic Remedies for Team Toxicity», *IEEE Software*, Julio de 1998, pp. 43-45.

- [KRA95] Kraul, R., y L. Streeter, «Coordination in Software Development», *CACM*, vol. 38, n.º 3, Marzo de 1995, pp. 69-81.
- [MAN81] Mantei, M., «The Effect of Programming Team Structures on Programming Tasks», *CACM*, vol. 24, n.º 3, Marzo de 1981, pp. 106-113.
- [PAG85] Page-Jones, M., *Practical Project Management*, Dorset House, 1985, p. VII.
- [REE99] Reel, J.S., «Critical Success Factors in Software Projects», *IEEE Software*, Mayo de 1999, pp. 18-23.
- [WEI86] Weinberg, G., *On Becoming a Technical Leader*, Dorset House, 1986.
- [WIT94] Whitaker, K., *Managing Software Maniacs*, Wiley, 1994.
- [ZAH94] Zahniser, R., «Timeboxing for Top Team Performance», *Software Development*, Marzo de 1994, pp. 35-38.

PROBLEMAS Y PUNTOS A CONSIDERAR

- 3.1. Basándose en la información contenida en este capítulo y en su propia experiencia, desarrolle «diez mandamientos» para potenciar a los ingenieros del software. Es decir, haga una lista con las diez líneas maestras que lleven al personal que construye software a su máximo potencial.
- 3.2. El Modelo de Madurez de Capacidad de Gestión de Personal (MMCGP) del Instituto de Ingeniería del Software hace un estudio organizado de las «áreas clave prácticas (ACP)» que cultivan los buenos profesionales del software. Su profesor le asignará una ACP para analizar y resumir.
- 3.3. Describa tres situaciones de la vida real en las que el cliente y el usuario final son el mismo. Describa tres situaciones en que son diferentes.
- 3.4. Las decisiones tomadas por una gestión experimentada pueden tener un impacto significativo en la eficacia de un equipo de ingeniería del software. Proporcione cinco ejemplos para ilustrar que es cierto.
- 3.5. Repase el libro de Weinberg [WEI86] y escriba un resumen de dos o tres páginas de los aspectos que deberían tenerse en cuenta al aplicar el modelo MOI.
- 3.6. Se le ha nombrado gestor de proyecto dentro de una organización de sistemas de información. Su trabajo es construir una aplicación que es bastante similar a otras que ha construido su equipo, aunque ésta es mayor y más compleja. Los requisitos han sido detalladamente documentados por el cliente. ¿Qué estructura de equipo elegiría y por qué? ¿Qué modelo (de proceso de software elegiría y por qué?
- 3.7. Se le ha nombrado gestor de proyecto de una pequeña compañía de productos software. Su trabajo consiste en construir un producto innovador que combine hardware de realidad virtual con software innovador. Puesto que la competencia por el mercado de entretenimiento casero es intensa, hay cierta presión para terminar el trabajo rápidamente. ¿Qué estructura de equipo elegiría y por qué? ¿Qué modelo(s) de proceso de software elegiría y por qué?
- 3.8. Se le ha nombrado gestor de proyecto de una gran compañía de productos software. Su trabajo consiste en dirigir la versión de la siguiente generación de su famoso procesador de textos. Como la competencia es intensa, se han establecido y anunciado fechas límite rígidas. ¿Qué estructura de equipo elegiría y por qué? ¿Qué modelo(s) de proceso de software elegiría y por qué?
- 3.9. Se le ha nombrado gestor de proyecto de software de una compañía que trabaja en el mundo de la ingeniería genética. Su trabajo es dirigir el desarrollo de un nuevo producto de software que acelere el ritmo de impresión de genes. El trabajo es orientado a I+D, pero la meta es fabricar el producto dentro del siguiente año. ¿Qué estructura de equipo elegiría y por qué? ¿Qué modelo(s) de proceso de software elegiría y por qué?
- 3.10. Como muestra la Figura 3.1, basándose en los resultados de dicho estudio, los documentos parecen tener más uso que valor. ¿Por qué cree que pasó esto y qué se puede hacer para mover el punto documentos por encima de la línea de regresión en el gráfico? Es decir, ¿qué se puede hacer para mejorar el valor percibido de los documentos?
- 3.11. Se le ha pedido que desarrolle una pequeña aplicación que analice todos los cursos ofrecidos por la universidad e informe de las notas medias obtenidas en los cursos (para un periodo determinado). Escriba una exposición del alcance que abarca este problema.
- 3.12. Haga una descomposición funcional de primer nivel de la función diseño de página tratado brevemente en la Sección 3.3.2.

OTRAS LECTURAS Y FUENTES DE INFORMACIÓN

Una excelente serie de cuatro volúmenes escrito por Weinberg (*Quality Software Management*, Dorset House, 1992, 1993, 1994, 1996) introduce los conceptos básicos sobre sistemas y conceptos de gestión, explica cómo usar mediciones eficazmente y menciona la «acción congruente», la habilidad de «encajar» las necesidades del gestor, del personal técnico y las del negocio. Proporciona información útil tanto a los gestores noveles como a los experimentados. Brooks (*The Mythical Man-Month*, Anniversary Edition, Addison-Wesley, 1995) ha actualizado su clásico libro para

proporcionar una nueva visión profunda de los aspectos del proyecto de software y de su gestión. Purba y Shah (*How to Manage a Successful Software Project*, Wiley, 1995) presentan un número de estudios de casos de proyectos que indican por qué unos fracasaron y otros fueron un éxito. Bennatan (*Software Project Management in a Client/Server Environment*, Wiley, 1995) estudia aspectos específicos de gestión asociados con el desarrollo de sistemas cliente/servidor.

Se puede argumentar que el aspecto más importante de la gestión del proyecto de software es la gestión de personal. El

libro definitivo sobre este tema lo escribieron DeMarco y Lister [DEM98], pero se han publicado en los últimos años los siguientes libros donde se examina su importancia:

Beaudouin-Lafon, M., *Computer Supported Cooperative Work*, Wiley-Liss, 1999.

Carmel, E., *Global Software Teams: Collaborating Across Borders and Time Zones*, Prentice Hall, 1999.

Humphrey, W. S., *Managing Technical People: Innovation, Teamwork, and the Software Process*, Addison-Wesley 1997.

Humphrey, W. S., *Introduction to the Team of Software Process*, Addison-Wesley, 1999.

Jones, P. H., *Handbook of Team Design: A Practitioner's Guide to Team Systems Development*, McGraw-Hill, 1997.

Karolak, D. S., *Global Software Development: Managing Virtual Teams and Environments*, IEEE Computer Society, 1998.

Mayer, M., *The Virtual Edge: Embracing Technology for Distributed Project Team Success*, Project Management Institute Publications, 1999.

Otro excelente libro de Weinberg [WEI86] es lectura obligada para todo gestor de proyecto y jefe de equipo. Le dará una visión interna y directrices para realizar su trabajo

más eficazmente. House (*The Human Side of Project Management*, Addison-Wesley, 1988) y Crosby (*Running Things: The art of Making Things Happen*, McGraw-Hill, 1989) proporcionan directrices prácticas para gestores que deban tratar con problemas humanos y técnicos.

Aunque no están relacionados específicamente con el mundo del software, y algunas veces sufren demasiadas simplificaciones y amplias generalizaciones, los libros de gran éxito de Drucker (*Management Challenges for the 21st Century*, Harperbusiness, 1999), Buckingham y Coffman (*First, Break All the Rules: What the World's Greatest Managers Do Differently*, Simon & Schuster, 1999) y Christensen (*The Innovator's Dilemma*, Harvard Business School Press, 1997) enfatizan «nuevas reglas» definidas por una economía que cambia con rapidez. Viejos títulos como *The One Minute Manager* e *In Search of Excellence* continúan proporcionando enfoques valiosos que pueden ayudarle a gestionar los temas relacionados con el personal de un modo más eficiente.

En Internet están disponibles una gran variedad de fuentes de información relacionadas con temas de gestión de proyectos de software. Se puede encontrar una lista actualizada con referencias a sitios (páginas) web que son relevantes para los proyectos de software en <http://www.pressman5.com>.