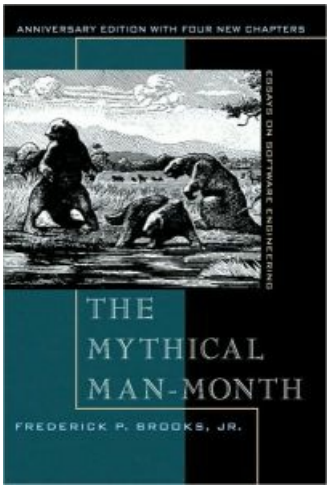

Libro El mítico mes-hombre

Ensayos sobre ingeniería de software

Frederick P. Brooks
Addison-Wesley, 1995
Primera Edición:1975
También disponible en: Inglés

Reseña

Este libro es un clásico por una razón. Todos los ensayos de Frederick P. Brooks Jr. son sobre ingeniería de software y demuestran ser invaluables para quienes se interesan en la historia y los procesos de ese campo. *BooksInShort*también recomienda este libro de Brooks a quien planee u organice proyectos importantes. La colección sigue siendo oportuna por la claridad de su pensamiento y el instruido encanto de su prosa. Cuando Brooks escribe sobre programación, nunca escribe sólo sobre programación. Escribe sobre las complejidades de la vida y sobre cómo planear, organizar y comunicar mejor los conceptos que necesita para superar esas complejidades. Esta edición del 20º. aniversario contiene nuevos ensayos en los que Brooks reflexiona sobre sus escritos anteriores – especialmente sus principios y pronósticos – y responde a sus críticos. El resultado exhibe cómo trabaja una mente singular y notablemente honesta.

Ideas fundamentales

- La integridad conceptual es la consideración más importante en el diseño de software.
- Una sola mente o unas cuantas personas de ideas afines deben encargarse de los proyectos complejos.
- La organización y documentación clara, completa y estandarizada son esenciales para cualquier esfuerzo de programación a gran escala.
- Agregar gente nueva al proceso de desarrollo frena el progreso.
- Los programadores son como los arquitectos: Su primera visión es clara; la segunda, demasiado complicada, y la tercera, mejor que la segunda.
- Comúnmente, los desastres no atrasan los proyectos. Los proyectos se atrasan día con día.
- Planee crear versiones múltiples de su producto.
- El desarrollo de software nunca crecerá tan rápida o constantemente como la capacidad de hardware.
- Cambie las metáforas que usa para pensar en programación; véala como “crecimiento” en vez de “construcción”.
- Para producir mejor software, aproveche las herramientas existentes y forme excelentes diseñadores de software.

Resumen

Los retos y placeres de la programación

¿Por qué le gusta a la gente programar? Porque le gusta hacer cosas, especialmente cosas útiles. Saborea su complejidad y el aprendizaje que conlleva. Aprecia la alegría de “trabajar en un medio tan maleable”. Como los poetas, los programadores trabajan con pensamiento casi puro. Pero, a diferencia de ellos, los pensamientos de los programadores crean cambios reales en el mundo físico. Como los magos, los programadores deben hacer su oficio perfectamente – y, aun así, los programas siempre tienen un sinnúmero de problemas. Además, cada vez que usted crea un programa que funciona bien, sabe que está condenado a pronto ser obsoleto, a medida que avanza la tecnología y surgen nuevos programas.

> “El costo, en efecto, varía de acuerdo con el número de hombres y el número de meses. El progreso, no. Por tanto el mes-hombre como unidad para medir el tamaño de un trabajo es un mito engañoso y peligroso”.

"La programación de grandes sistemas" incluye complejidades especiales, pero "los programadores son optimistas" y suponen que todo saldrá bien. Además, la gente tiene el malentendido común y fundamental de cómo funciona el trabajo. La medición llamada "mes-hombre", que mide el tiempo según el número de personas que se requieren durante un número de meses para completar una tarea, crea suposiciones incorrectas y peligrosas. Postula que "los hombres y los meses son intercambiables", es decir, que los administradores de proyectos pueden dividir el trabajo perfectamente, que cualquiera puede hacer el trabajo y que nadie necesita comunicarse. Esto podría ser cierto para el trabajo burdo, como "cosechar trigo", pero es definitivamente falso para la programación o cualquier tarea intelectualmente precisa. Hay tareas que no se pueden dividir. Otras sí, pero requieren procesos sofisticados de comunicación para enseñar a la gente cómo hacer las tareas, organizarlas, sintetizar sus resultados y demás. Como resultado, agregar más gente a un proyecto agrega trabajo – trabajo difícil y que lleva tiempo – y, por tanto, hay más gente que se toma más tiempo para completar esas tareas, no menos.

> "La Torre de Babel fue, tal vez, el primer fiasco ingenieril, pero no el último. La comunicación y su resultado, la organización, son críticas para el éxito".

El tamaño de la tarea presenta otro reto: encontrar formas de organizar a sus equipos para hacer el mejor uso del personal. Una táctica se inspira en un grupo quirúrgico y proporciona tareas específicas a todos para apoyar al "jefe de programación", como los enfermeros en un quirófano ayudan a un cirujano. Reclute gente para tareas específicas y organícela para crear, proteger y mantener la claridad intelectual. El "copiloto" se une al jefe de programación como una caja de resonancia "menos experimentada" que entiende el programa y al programador, y a menudo representa al jefe ante el equipo. El "administrador" mantiene las cosas organizadas y funcionando; el "editor" genera y mantiene el documento real que el equipo crea. Los secretarios y otros empleados trabajan bajo el mando del editor, y todos colaboran de cerca con el "abogado del lenguaje", quien sigue las reglas y peculiaridades específicas del lenguaje de programación. Un "herramientero" crea herramientas (como "funciones especializadas") a medida que las necesita el cirujano, y el "probador" desarrolla "casos de prueba adecuados" para probar el programa.

> "La integridad conceptual es la consideración más importante en el diseño de sistemas".

A algunas catedrales europeas no se les dio continuidad de una parte a otra, porque las diversas generaciones de trabajadores veían el proyecto complejo de forma distinta. Sin embargo, la catedral de Reims tiene una asombrosa y extraña "unidad arquitectónica". Un gran programa requiere una "integridad conceptual" similar – el único y más importante factor para diseñar un sistema. Esta integridad conceptual significa que una sola mente, o un "número muy pequeño" de mentes que está de acuerdo, está al mando. Esto lleva a una aristocracia necesaria en el diseño de software, en la que los pocos visionarios están al mando. Esto no es absoluto, pues la idea creativa debe implementarse y podría requerir más gente. La información siempre debe fluir del proceso de implementación de vuelta a los diseñadores, quienes entonces toman en cuenta la nueva información. Los sistemas computacionales pasan por "tres fases distintas: arquitectura, implementación y realización". Aunque la mayoría de las artes pasa por estas etapas secuencialmente, usted puede, en diseño de software, pasar por ellas simultáneamente.

Cuestiones organizacionales

Los arquitectos siguen patrones similares, ya sea que trabajen con mármol o con un lenguaje de programación. La primera versión de cualquier sistema o diseño tiende a ser "sobria y clara". Son cuidadosos al diseñar con un presupuesto limitado, y a sabiendas de que no saben totalmente lo que quieren hacer. Por eso, cuando surgen ideas durante la fase inicial, la gente las deja "para la próxima vez". Luego, la segunda versión está atascada de excesos. Para la tercera versión, los arquitectos ya saben qué funciona y se limitan a eso.

> "La primera obra de un arquitecto tiende a ser sobria y clara".

Cuando diseñe un proyecto de software, ciertas formas de documentación y comunicación son esenciales. Necesita un manual que describa todo lo que el usuario ve, pero nada que el usuario no vea. El estilo debe estar claro y unificado. Las definiciones deben ser inusualmente específicas y precisas, y deben aclarar cualquier confusión o ambigüedad. A medida que avanza, hable del proyecto en dos tipos de reuniones. Primero, haga reuniones semanales para actualizar a todos. Funcionan mejor si sólo asiste la gente que trabaja en el proyecto (sin consultores ni observadores). Antes de la reunión, envíe por escrito los cambios planeados y sea flexible respecto a la resolución de problemas, pero sea claro sobre quién toma qué decisiones. En el segundo tipo de reunión, una sesión anual extendida, resuelva problemas menores y desacuerdos que se acumulan con el tiempo. Documente las decisiones importantes y asegúrese de que todos sepan quién tiene los documentos oficiales actualizados. Registre todas las llamadas telefónicas y haga circular los registros de todas las llamadas que den información importante de los arquitectos principales.

Torre de claridad

La comunicación es el componente más crítico de los proyectos complejos de programación. Un proyecto es como la Torre de Babel: Con comunicaciones claras y unificadas, uno puede llegar al cielo, pero si se interrumpen las comunicaciones, se caerá la edificación.

> "Más allá de la artesanía se encuentra la invención, y es ahí donde nacen los programas sobrios, eficientes y rápidos".

Si es usted un nuevo administrador de proyectos, puede ver todo el papeleo que se requiere como un obstáculo y pérdida de tiempo que no le permite hacer el trabajo real. Al ir adquiriendo experiencia, entenderá la necesidad de la documentación. Los registros cuidadosos revelan los "huecos" y las "inconsistencias" en su pensamiento. Un registro escrito facilita la comunicación y le permite transformar sus impulsos internos en listas de verificación que se pueden organizar. Documentar los programas mejora la manera de pensar en ellos. Al documentar los proyectos, puede ver que hay elementos específicos de su documentación que son comunes a todos los proyectos. Ya sea que diseñe software u organice un departamento universitario, necesita expresar categorías similares. Empiece con los "objetivos", luego pase a las "especificaciones".

> "La tendencia general es diseñar de más el segundo sistema, usando todas las ideas y adornos que cuidadosamente se habían dejado de lado en el primero".

Como administrador de proyectos, necesita un "cuaderno de proyectos" que contenga y organice todos los elementos principales del proyecto: objetivos, estándares, memoranda principales y demás. En programación, este cuaderno se convierte en su "manual y en las especificaciones de desempeño". Los programas de todo tipo requieren cronogramas y presupuestos; necesitan una tabla organizacional que represente cómo se relacionan las personas que trabajan juntas, y una "asignación de

espacios" que muestre dónde estarán y cuánto espacio ocuparán. Al introducir un nuevo programa, calcule lo que requerirá, prediga cuándo y cómo lo completará y ponga precio a su oferta para un mercado identificado. El tiempo y tamaño del programa son dos importantes componentes del costo. Generar tablas o gráficos que representen sus datos lo ayudará a tomar mejores decisiones sobre las compensaciones.

> "Un control riguroso durante las pruebas es una de las impresionantes técnicas de depuración de hardware, y eso se aplica también a los sistemas de software".

A veces los proyectos se atrasan y no cumplen con su fecha límite. Generalmente no se debe a un gran desastre, sino a pequeñas fallas aquí y allá. Para evitar una acumulación de atrasos, defina sus "puntos de referencia" con "gran agudeza mental". Entre más concretos y específicos sean los puntos de referencia, más fácil será verificar su terminación. Organícelos en una gráfica de Evaluación de Programas y Técnica de Revisión o un "cronograma de ruta crítica" para saber qué elementos retrasarán acciones futuras. Como líder, facilite la divulgación a sus subordinados y disciplínese para dejarlos manejar sus propios retos. Actualice su gráfica de seguimiento con sesiones regulares de información. Al reunir a su equipo, busque lo que los atletas llaman "empuje". La disposición de los miembros de equipo para esforzarse y trabajar más rápido cuando afrontan una crisis compensará los atrasos menores.

Planear para el cambio y la transformación

La primera versión de lo que haga será torpe, ineficiente y "apenas utilizable". Por tanto, planee esta versión "desechable" de su programa o modelo, y ajuste sus cronogramas y presupuestos de manera correspondiente. Esta práctica de reunir y luego desechar es parte de un cambio mayor que se requiere en la organización: No sólo está cambiando lo que hace o produciendo nuevas versiones. Debe diseñar el sistema entero para que se adapte y cambie, así que estructure su organización de manera acorde. Esta restructuración será difícil.// // Como administrador de proyectos, usted debe satisfacer las exigencias de su proyecto (y cambiar programadores de acuerdo con ello), tomar en cuenta el deseo de los individuos de aprender y crecer, y satisfacer la necesidad de prestigio, reconocimiento y conocimiento de la gente dentro de su lugar en la jerarquía. Bell Labs afronta estos problemas deshaciéndose de los "títulos del cargo". IBM establece una "escala doble de avance" para que la gente ascienda por una escalera técnica o administrativa en la que todos los "peldaños" sean igual de gratificantes. Su estructura debe poder adaptarse desde la creación de un producto específico hasta su conservación. "El costo total de mantener un programa ampliamente usado es típicamente el 40% o más del costo de desarrollarlo".

> "Los defectos más perjudiciales y sutiles son los defectos de sistemas, que surgen de suposiciones desiguales de los autores de diversos componentes".

Considere en sus planes el ciclo específico de defectos y problemas que desarrollarán sus programas de software. Los nuevos programas generan muchos defectos al principio, luego menos y más tarde vuelve a aumentar su número, reflejando los nuevos defectos que ocurren en un nivel más sutil. Puede resolver los defectos de varias maneras. Al nivel de diseño, su meta debe ser la integridad conceptual. Piense en su programa como una obra arquitectónica, y evalúe cualquier cambio planeado en términos de las funciones que afecta. Diseñe de arriba hacia abajo, comenzando con el nivel más alto de "refinamiento". Pruebe las especificaciones antes de escribir cada unidad de código; después pruebe cada nivel del programa, también de arriba hacia abajo. Programe "y depure en un lenguaje de alto nivel" que permita depuraciones interactivas y en tiempo real. Asigne suficiente tiempo para la depuración; en la mayoría de los casos, requerirá tanto tiempo como escribir el programa. Una vez que el sistema esté construido, puede depurar desde otros ángulos. "Use componentes depurados" si puede, agregándolos de uno en uno al sistema. Cree un "componente de prueba" y un "archivo miniatura" sólo con el propósito de hacer pruebas.

> "El programador desesperado por falta de espacio a menudo trabaja mejor si se aleja de su código, retrocede y contempla su información. La representación es la esencia de la programación".

Cualquier programa requiere una documentación clara y completa. Sin embargo, no puede sólo insistir en que su personal la proporcione. Primero, describa el programa, qué hace y cómo funciona. Identifique el hardware necesario, las decisiones que deben tomar los usuarios y cómo deben ingresar los datos. Una visión general del programa es esencial. Puede utilizar un diagrama de flujo de una página que documente la "estructura de decisiones" del programa. Adhiérase a anotaciones estándar para todos los componentes de un programa.

Cumplir con los retos fundamentales del software: Sin "solución garantizada"

La gente se ha acostumbrado a las mejoras de hardware que hacen que las computadoras sean cada vez más rápidas. Esto la lleva a esperar una solución garantizada que disipe las "dificultades esenciales" implicadas en el desarrollo de software. Algunos recurren a la inteligencia artificial para que proporcione esta herramienta mágica; otros, a la "programación automática" o "programación gráfica", y otros más, a los "sistemas expertos". Aunque los avances en cada una de estas áreas son reales, ninguno cumple con las expectativas de los creyentes. Debido a la naturaleza de las dificultades de la programación, no hay una solución garantizada.

> "La tecnología, la organización que la rodea y las tradiciones del oficio conspiran para definir cierta cantidad de papeleo que debe prepararse para un proyecto".

Mientras los diseñadores de hardware afrontan los retos del universo físico, los diseñadores de software deben lidiar con la "complejidad arbitraria" de la cultura humana. El software requiere actualizaciones continuas y, aunque puede ser representado de varias formas, es fundamentalmente intangible y tal vez imposible de visualizar. En el pasado, los mayores avances vinieron de "lenguajes de alto nivel", de "tiempo compartido" y de la creación de "entornos de programación unificada". Más recientemente los avances se han desarrollado con base en la posibilidad de comprar software, más que crearlo, y en la reutilización de componentes. Ayudar a los clientes a determinar sus necesidades y brindar "prototipos rápidos" será esencial en el futuro. Aunque la "programación orientada al objeto" no se ha desarrollado tan rápidamente como algunos seguidores hubieran esperado, también ofrece posibilidades.

> "Ley de Brooks: Agregar recursos humanos a un proyecto de software retrasado lo atrasa más".

Para tener éxito, recurra a dos enfoques que reconocen los retos humanos y artísticos esenciales del software. Primero, cambie las metáforas. En el pasado, los programadores hablaban comúnmente de escribir programas. Luego surgió la idea de "construirlos", que proporcionó una gran cantidad de metáforas relacionadas: "andamios" conceptuales, "ensamblaje de componentes" y otras. Esa mentalidad está llegando al fin de su utilidad. Ha llegado la hora de pensar en cultivar programas,

y de hacerlo en aumento. Reconozca que el diseño de software es un arte, y que el arte avanza mediante el trabajo de grandes artistas; por tanto, “cultive grandes diseñadores”. Encuentre individuos talentosos desde el inicio, y ayúdelos a fraguar carreras profesionales que les den una experiencia amplia y profunda. Genere lugares de encuentro donde puedan interactuar con otros “diseñadores en formación” y proporcióneles asesores que los ayuden a desarrollarse.

Sobre el autor

Frederick P. Brooks Jr. es el autor de *The Design of Design* y otras obras. Ganó el Premio A.M. Turing de la Association for Computing Machinery en 1999.