

INTRODUCCIÓN AL PROCESO SOFTWARE PERSONALSM

CONSULTORES EDITORIALES:

SEBASTIÁN DORMIDO BENCOMO
Departamento de Informática y Automática
UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

LUIS JOYANES AGUILAR
Departamento de Lenguajes, Sistemas Informáticos e
Ingeniería del Software
UNIVERSIDAD PONTIFICIA DE SALAMANCA en Madrid

INTRODUCCIÓN AL PROCESO SOFTWARE PERSONALSM

WATTS S. HUMPHREY
CARNEGIE MELLON UNIVERSITY

Traducción:

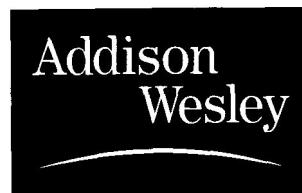
Javier Zapata Martínez
Universidad de Murcia

Coordinador de la traducción:

Jesús García Molina
Universidad de Murcia

Revisión Técnica:

José Antonio Cerrada Somolinos
Universidad Nacional de Educación a Distancia



Madrid • México • Santafé de Bogotá • Buenos Aires • Caracas • Lima • Montevideo •
San Juan • San José • Santiago • São Paulo • Reading, Massachusetts • Harlow, England

/ Datos de catalogación bibliográfica

Humphrey W.S.
Introducción al Proceso Software PersonalSM
PEARSON EDUCACIÓN, S.A., Madrid, 2001

ISBN 84-7829-052-4
Materia: Informática: 681.3

Formato: 170 x 240

Páginas: 328

Watts. S. Humphrey

Introducción al Proceso Software PersonalSM

No está permitida la reproducción total o parcial de esta obra ni su tratamiento o transmisión por cualquier medio o método sin autorización escrita de la Editorial.

DERECHOS RESERVADOS

© 2001 respecto a la primera edición en español por:
PEARSON EDUCACIÓN, S.A.
Núñez de Balboa, 120
28006 Madrid

ISBN: 84-7829-052-4

Depósito Legal: M- Impreso por: Sprint, S.L.

ADDISON WESLEY es un sello editorial autorizado de PEARSON EDUCACIÓN, S. A.

Traducido de:

Introduction to the Personal Software ProcessSM, First Edition by Watts Humphrey
© 1997, por Pearson Education, S. A.
ISBN: 0-201-54809-7

Edición en español:

Equipo editorial:

Editor: Andrés Otero

Asistente editorial: Ana Isabel García

Equipo de producción:

Director: José A. Clares

Técnico: José A. Hernán

Diseño de cubierta: Mario Guindel y Yann Boix

Composición; DiScript Preimpresión, S. L.

Impreso por: Depósito Legal: M-28.980-2001'

IMPRESO EN ESPAÑA - PRINTED IN SPAIN

Este libro ha sido impreso con papel y tintas ecológicos

A BÁRBARA

Mi amor, mi esposa, mi vida

Contenido

<i>Prólogo de los profesores</i>	XV
<i>Prólogo de los estudiantes</i>	XIX
<i>Prólogo</i>	XXI
1 <i>El trabajo del ingeniero del software</i>	1
1.1 ¿Qué es la ingeniería del software?.....	1
1.2 ¿Por qué es importante una buena ingeniería?.....	2
1.3 El proceso personal del software	2
1.4 La disciplina del trabajo de alta calidad.....	3
1.5 La importancia del trabajo de alta calidad	4
1.6 ¿Cómo mejorar la calidad de tu trabajo?	4
1.7 El proceso de mejora	5
1.8 El objetivo de este libro	6
Resumen	7
Ejercicio 1	8
Referencia.....	8
2 <i>La gestión del tiempo</i>	9
2.1 La lógica de la gestión del tiempo	9
2.2 Comprende cómo utilizas el tiempo	11
2.3 El cuaderno de ingeniería.....	12

2.4	El diseño del cuaderno	12
2.5	Ejemplos de cuadernos de ingeniería.....	14
	Resumen	16
	Ejercicio 2	16
3	<i>El control del tiempo</i>	17
3.1	¿Por qué controlar el tiempo?	17
3.2	El registro de los datos de tiempos.....	18
3.3	El control del tiempo	19
3.4	El uso de una tabla de registro de tiempos normalizada	20
3.5	La gestión de las interrupciones.....	22
3.6	El control de las tareas finalizadas	24
3.7	¿Cómo mantener la tabla de tiempos en el cuaderno de ingeniería? ..	25
3.8	Ideas para registrar tu tiempo	25
	Resumen	27
	Ejercicio 3	27
4	<i>Planificación de períodos y productos</i>	29
4.1	Planes de períodos y productos	29
4.2	El resumen semanal de actividades.....	31
4.3	El resumen de los tiempos semanales	33
4.4	Cálculo de los tiempos y medias del período.....	35
4.5	La utilización del resumen semanal de actividades	38
	Resumen	41
	Ejercicio 4	41
5	<i>La planificación del producto</i>	43
5.1	La necesidad de los planes del producto	43
5.2	Por qué son útiles los planes del producto	44
5.3	¿Qué es un plan del producto?	45
5.4	La planificación del producto en este libro	45
5.5	La planificación de pequeños trabajos	46
5.6	Algunas definiciones.....	46
5.7	El Cuaderno de Trabajos	46
5.8	Algunas sugerencias sobre cómo utilizar el Cuaderno de Trabajos ..	49
5.9	La utilización de tiempos y medias del producto	54
	Resumen	55
	Ejercicio 5	55

6	<i>El tamaño del producto</i>	57
6.1	El proceso de planificación del producto	57
6.2	Medida del tamaño	58
6.3	Algunas precauciones sobre la utilización de las medidas de tamaño	58
6.4	El tamaño de un programa	59
6.5	Otras medidas del tamaño	62
6.6	La estimación del tamaño del programa	62
6.7	Cómo hacer estimaciones de tamaños mayores	64
6.8	Cómo utilizar medidas de tamaño en el Cuaderno de Trabajos	65
	Resumen	69
	Ejercicio 6.....	71
7	<i>La gestión del tiempo</i>	73
7.1	Elementos de la gestión del tiempo.....	13
7.2	La clasificación de las actividades	74
7.3	La recogida de datos sobre el tiempo dedicado a cada actividad....	74
7.4	Cómo evaluar tu distribución del tiempo	74
7.5	Cómo hacer una estimación de tiempo	75
7.6	Cómo encontrar más tiempo	77
7.7	Cómo establecer reglas básicas	77
7.8	Cómo priorizar tu tiempo	79
7.9	La gestión del tiempo estimado	82
7.10	Sugerencias para la gestión del tiempo variable	84
7.11	Tu objetivo en la gestión del tiempo	85
	Resumen	85
	Ejercicio 7.....	86
8	<i>La gestión de los compromisos</i>	87
8.1	Definición de compromiso	87
8.2	Responsabilidad para hacer compromisos	89
8.3	Ejemplo de un compromiso	90
8.4	Un ejemplo en la industria	91
8.5	La gestión de compromisos no conseguidos	92
8.6	La importancia de gestionar compromisos	93
8.7	Las consecuencias de no gestionar los compromisos	94
8.8	La forma de gestionar compromisos	95
	Resumen	96
	Ejercicio 8.....	96

9 *La gestión de las programaciones* **99**

9.1	La necesidad de las programaciones	99
9.2	El diagrama de Gantt.....	100
9.3	Cómo hacer una programación de un proyecto	101
9.4	Puntos de control	103
9.5	El seguimiento de los planes de los proyectos	105
9.6	El seguimiento del valor conseguido	108
	Resumen	112
	Ejercicio 9.....	112
	Referencias	112

10 *El plan del proyecto* **113**

10.1	La necesidad de los planes de los proyectos	113
10.2	El Resumen del Plan del Proyecto	114
10.3	El Resumen	116
10.4	El tamaño del programa	117
10.5	El tiempo en Fases	120
10.6	Cómo estimar con exactitud.....	122
	Resumen	122
	Ejercicio 10.....	122
	Referencias	123

11 *El proceso de desarrollo del software* **125**

11.1	Por qué utilizamos los procesos	125
11.2	Algunas definiciones	126
11.3	El guión del proceso	127
11.4	Puntos de control y fases	129
11.5	Actualización de la tabla Resumen del Plan del Proyecto	130
11.6	Un ejemplo de planificación	132
11.7	Un ejemplo para calcular los datos hasta la fecha	135
	Resumen	138
	Ejercicio 11.....	138
	Referencia.....	139

12 *Defectos* **141**

12.1	¿Qué es la calidad del software?	141
12.2	Defectos y calidad	142
12.3	¿Qué son los defectos?	143

12.4	Defectos versus bugs	144
12.5	Tipos de defectos	146
12.6	La comprensión de los defectos	147
12.7	El cuaderno de registro de defectos	148
12.8	La contabilización de los defectos	153
12.9	La utilización del cuaderno de registro de defectos	154
12.10	El proceso del PSP actualizado	155
	Resumen	159
	Ejercicio 12.....	161
	Referencias	161

i3 Encontrar defectos**i63**

13.1	Un compromiso personal con la calidad	163
13.2	Los pasos para encontrar defectos	163
13.3	Formas de encontrar y corregir defectos	164
13.4	La revisión del código	166
13.5	¿Por qué hay que encontrar pronto los defectos ?	167
13.6	El coste de encontrar y corregir defectos	167
13.7	El uso de las revisiones para encontrar defectos	169
13.8	Revisar antes de compilar	170
13.9	Datos sobre defectos de compilación y pruebas	172
13.10	Actualización de la tabla Resumen del Plan del Proyecto del PSP	174
13.11	Otras clases de revisiones.....	174
	Resumen,	178
	Ejercicio 13.....	180
	Referencias	180

**i4 Listas de comprobación para la revisión
del código****i83**

14.1	¿Por qué ayudan las listas de comprobación?	183
14.2	Un ejemplo de lista de comprobación para la revisión de código...	184
14.3	Utilización de una lista de comprobación para la revisión de código	187
14.4	La elaboración de una lista de comprobación personal	188
14.5	La mejora de la lista de comprobación	194
14.6	Estándares de codificación	196
	Resumen	197
	Ejercicio 14.....	200

15	<i>La previsión de defectos</i>	201
15.1	El porcentaje de defectos	201
15.2	La utilización de los datos de defectos.....	203
15.3	Densidad de defectos	204
15.4	Cómo estimar las tasas de defectos	204
15.5	Estimación de defectos.....	205
15.6	Actualización y ejemplo de la tabla Resumen del Plan del Proyecto	207
15.7	La anotación de los datos reales.....	212
	Resumen	214
	Ejercicio 15.....	215
	Referencia	215
16	<i>La economía de eliminar defectos</i>	217
16.1	La necesidad del trabajo de calidad	217
16.2	El problema de eliminar defectos.....	218
16.3	El tiempo de eliminar defectos.....	219
16.4	Experiencia en la introducción y eliminación de defectos	219
16.5	Ahorro en la eliminación de defectos	221
16.6	El cálculo de los Defectos/Hora en el Resumen del Plan del Proyecto del PSP	223
16.7	El cálculo del rendimiento en el Resumen del Plan del Proyecto ...	225
16.8	La mejora de las tasas de eliminación de defectos	229
16.9	La reducción de las tasas de introducción de defectos	230
	Resumen	231
	Ejercicio 16.....	232
	Referencias	232
17	<i>Defectos de diseño</i>	233
17.1	La naturaleza de los defectos de diseño	233
17.2	Identificación de los defectos de diseño.....	235
17.3	¿Qué es diseñar?.....	235
17.4	El proceso de diseño	236
17.5	Las causas de los defectos del diseño	237
17.6	El impacto de los defectos de diseño	238
17.7	Representación del diseño	239
	Resumen	244
	Ejercicio 17.....	245
	Referencias	246

18	<i>Calidad del producto</i>	247
18.1	La calidad viene primero	247
18.2	Las pruebas	248
18.3	El filtro de las pruebas	249
18.4	Los beneficios del trabajo cuidadoso	251
18.5	El cálculo de los valores de rendimiento	251
18.6	La estimación del rendimiento definitivo	254
18.7	Los beneficios de un rendimiento de proceso del 100%	256
18.8	Experiencia del rendimiento	257
18.9	Prototipado	258
	Resumen	258
	Ejercicio 18.....	259
	Referencia.....	259
19	<i>La calidad del proceso</i>	261
19.1	Medidas del proceso	261
19.2	La paradoja de la eliminación de defectos	262
19.3	Una estrategia para la eliminación de defectos	263
19.4	El coste de la calidad	264
19.5	Cálculo del coste de la calidad	265
19.6	La relación Valoración/Fallos	267
19.7	Cómo mejorar las tasas de revisión	273
19.8	Cálculo del verdadero Coste de Calidad	275
	Resumen	276
	Ejercicio 19.....	277
20	<i>Un compromiso personal con la calidad</i>	279
20.1	La importancia de la calidad	279
20.2	El aumento del riesgo asociado a la poca calidad	280
20.3	Cómo hacer un compromiso con la calidad	282
20.4	Tus objetivos personales	282
20.5	Las recompensas del logro	283
	Referencia	284
	<i>Índice analítico</i>	285
	<i>Páginas suplementarias</i>	293

Prólogo de los profesores

Nosotros utilizamos un borrador de este libro para enseñar los principios de los procesos en el primer curso del programa de Informática, en la Universidad Aeronáutica de Embry-Riddle. El libro proporciona un subconjunto de los elementos y actividades del Proceso Software Personal (PSP)SM, acrónimo de Personal Software Process, que un estudiante de primer año puede fácilmente asimilar junto con los temas más tradicionales del primer año de programación. Este libro también proporciona la motivación y una estructura para introducir a los estudiantes en prácticas personales disciplinadas. Hemos disfrutado utilizando este libro y comprobado cómo ayuda a nuestros estudiantes, para llegar a ser profesionales competentes del software.

Durante años, hemos estado intentando proporcionar a nuestros estudiantes experiencias reales de ingeniería del software. Hemos tenido un éxito moderado con la introducción de la teoría y práctica de la ingeniería del software al principio del currículum, y añadiendo proyectos en equipo en cursos superiores. Desafortunadamente, hemos encontrado que cuando los estudiantes trabajan en estos proyectos, no entienden la gestión del tiempo, la planificación y la gestión de la calidad. La industria encontró que la capacidad de los equipos de ingeniería para desarrollar productos software de calidad de una forma eficiente y eficaz, dependía en gran parte de la capacidad individual de los ingenieros. Reflexionando sobre ello, los problemas de los estudiantes con la gestión del tiempo y de la calidad no nos sorprendieron ya que no se les había dado cursos sobre cómo planificar y gestionar su trabajo. Decidimos intentar introducir los conceptos sobre procesos al comienzo del plan de estudios universitarios.

SM Personal Software Process y PSP son marcas registradas de la Universidad de Carnegie Mellon.

Observamos que los estudiantes de primer curso podían aprender mejor y beneficiarse más directamente de las prácticas de gestión del tiempo, por ello comenzamos introduciendo del Capítulo 1 al Capítulo 10 de este libro asignaturas del CS1¹. Aunque todos los estudiantes que accedieron a nuestros cursos del CS 1 tenían alguna experiencia en programación, no estaban aún preparados para un proceso de desarrollo de software formalmente definido. Necesitaban enfrentarse antes a los problemas de desarrollo de software para que pudieran comprender los papeles y prácticas de los ingenieros de software.

Después de finalizar el CS1 y completar su primer semestre, los estudiantes estaban preparados para desarrollar programas de una forma más disciplinada. Entonces introdujimos el proceso PSP en el CS2², utilizando el material de los Capítulos 11 al 20. Aquí, el estudiante planificó cada uno de sus proyectos de programación. Siguiendo las prácticas definidas en el PSP, utilizaron sus propios datos históricos para estimar el tamaño, el esfuerzo y la calidad (estimación de defectos). También reunieron y registraron los datos reales de cada proyecto en una tabla resumen.

Después de un año de experiencia, hemos encontrado que el enfoque de introducir las actividades de los procesos en los primeros cursos de CS, permite trabajar a los estudiantes. Con “permite trabajar” queremos decir que los estudiantes pueden aprender cómo utilizar los procesos tratados en este libro. Con el tiempo ven el valor de registrar los datos de esfuerzo, tamaño y calidad, y pueden utilizar estos datos en la planificación de proyectos y analizar su efectividad personal. La recogida de datos de su propio trabajo les da unas bases cuantitativas para la estimación. Regularmente realizan revisiones estructuradas y aprenden las fases de desarrollo definidas en su trabajo (por ejemplo, planificación, diseño, codificación, compilación, pruebas y postmortem). También observamos que el retraso en la introducción del PSP para otro semestre (*o año*) llevaría consigo que los alumnos adquiriesen prácticas de programación descuidadas e indisciplinadas bien arraigadas y serían más resistentes al cambio.

El PSP ha ayudado a los estudiantes a entender la importancia de un enfoque disciplinado al desarrollo del software. También proporciona un fundamento más riguroso para la posterior introducción de cuestiones más avanzadas de tipo individual y de equipo. La mayor parte de los datos de los estudiantes son exactos, pero hay que ser cuidadoso al analizar y rechazar los datos sospechosos. Desafortunadamente, los estudiantes no

¹ Asignatura de programación del primer semestre del primer curso en los currícula USA (N. del T.).

² Asignatura de programación del segundo semestre del primer curso en los currícula USA (N. del T.).

mejoraron en la planificación de su trabajo. Muchos aplazaron sus ejercicios hasta que no tuviesen los datos adecuados: problema perpetuo en los inicios de los programadores.

No fue sorprendente descubrir que el éxito de la técnica del PSP dependía en gran medida de nuestra capacidad para motivar a los estudiantes a aprender y practicar estos conceptos. Utilizamos las ideas y argumentos de este libro para animar a tener una visión positiva de los métodos del proceso. Observamos, que facilitando la retroalimentación en la clasificación y análisis de los datos, se estimulaba a los estudiantes a interesarse en mayor medida por sus datos personales. También fue útil invitar a los profesionales de la industria a discutir en clase sus experiencias en los procesos.

Hubo algunos problemas al comenzar la enseñanza en los nuevos cursos. Inicialmente, no teníamos suficientemente integradas las materias del PSP con el resto de materias de CS1 y CS2. Los estudiantes tuvieron problemas al relacionar las actividades de gestión del tiempo con sus trabajos de programación. Nosotros también fallamos en proporcionarle suficiente retroalimentación en los datos acumulados de las clases.

Un efecto interesante y beneficioso del PSP fue la gran cantidad de datos suministrados al profesor. En el CS1 hacemos resúmenes semanales de actividades sobre como los estudiantes gastan su tiempo en el curso. En el CS2 hacemos un resumen del PSP sobre cada proyecto de programación y obtenemos datos de tamaño, esfuerzo y defectos. A menudo estos datos provocan discusiones sobre los métodos enseñados en el curso y cómo afectan a la productividad del programador y a la calidad del programa. El PSP proporciona una base cuantitativa para el análisis y discusión de dichas cuestiones.

Continuamos enseñando el PSP a los estudiantes del primer año de nuestro programa. También requerimos a los estudiantes que han completado el CS1 y el CS2 a que utilicen el PSP en los cursos de algoritmos y estructuras de datos que siguen al CS2. Creemos que estarán mejor preparados para trabajar en equipos de proyectos complejos en su futuro profesional. También pretendemos orientar a los estudiantes a extender y mejorar el PSP en cursos posteriores.

Hemos encontrado este libro útil para introducir a nuestros estudiantes en la disciplina del software profesional y esperamos que otros estudiantes y profesores que utilicen este libro tengan experiencias similares a la nuestra.

Thomas B. Hilburn, Aboalfazl Salimi, Massood Towhidnejad
Universidad Aeronaútica de Enzby-Riddle

Prólogo de los estudiantes

Después de acabar el curso del **PSP** en el primer año, los profesores de la facultad de la Universidad Aeronáutica de Embry-Riddle nos preguntaron si nos gustaría colaborar en la redacción del prólogo del libro de texto. Nosotros aceptamos. Puesto que no sabíamos cómo escribir un prólogo, nos sugirieron que sencillamente respondiésemos a algunas preguntas.

He aquí las preguntas y las respuestas:

1. ¿Qué tipo de tareas hicisteis en el curso del PSP?

Registramos todo el tiempo que dedicábamos a los ejercicios de programación y a los proyectos. Había mucha documentación. También registramos el tamaño del programa y los defectos cometidos, y utilizamos los datos recogidos para estimar el tiempo, el tamaño y los defectos de futuros proyectos.

2. ¿Cómo hicisteis el trabajo? ¿Cómo adaptasteis este material a los materiales de los otros cursos?

Lo adaptamos bien al trabajo del curso, y disponiendo de una estimación nos ayudó a coger confianza en lo que estábamos haciendo.

Al principio, parece que el trabajo del **PSP** es un obstáculo a los otros trabajos del curso, pero una vez que llegas al final del curso, te das cuenta de que estas actividades realmente te ayudan a completar tu trabajo. Al avanzar el curso te preguntas: “¿por qué estoy haciendo esto?”, pero más tarde comienzas a ver que disponer de una estimación de lo que estás haciendo te ayuda realmente a completar el programa.

Es muy importante no amañar los datos (tiempos) porque entonces no te serán tan útiles.

3. ¿Qué aprendisteis?

Además de lo que ya hemos dicho, aprendes cómo puedes utilizar tu tiempo más eficientemente y a trabajar antes de utilizar el computador. Terminas haciendo mucho trabajo sobre papel antes de utilizar el ordenador.

Aprendes sobre tus errores y sobre los de otras personas (por medio de ejemplos y discusiones). También te ayuda a organizar tu programación (puesto que haces el trabajo en papel antes de utilizar el computador). El PSP también se puede utilizar en otras actividades (no solamente en el desarrollo del software), aunque es necesario modificar las tablas.

4. ¿Qué recomendaríais a otros estudiantes que van a utilizar el PSP en el futuro?

Hacerlo bien. No amañar los datos. Seguir las instrucciones. Intentar entender el marco de referencia y sus conceptos. No permitir que el papel les coma, los liquidará.

Ben Bishop, Andrew Henderson, Michael Patrick
Universidad Aeronáutica de Embry-Riddle

Prólogo

Si estás estudiando para ser un ingeniero de software, este libro está diseñado para ti. Describe los métodos que muchos ingenieros de software experimentados utilizan para hacer un trabajo competente, y proporciona ejercicios para ayudarte a aprender estos métodos. Cada capítulo describe un tema sencillo que será practicado cuando hagas los ejercicios. Los ejemplos de cada ejercicio te ayudarán a comprobar tu trabajo.

¿POR QUÉ HE ESCRITO ESTE LIBRO?

El desarrollo de productos software implica algo más que escribir instrucciones de programación juntas y ejecutarlas en un ordenador. Requiere cumplir los requisitos del cliente a un coste y planificación acordada. Para tener éxito, los ingenieros de software necesitan producir de forma regular programas de alta calidad de acuerdo con una planificación y unos costes. Este libro muestra cómo hacerlo. Te introduce al Proceso Software Personal (PSP), que es una guía que utiliza prácticas personales disciplinadas para ser un ingeniero de software.

El PSP mostrará cómo planificar y revisar tu trabajo, y cómo producir de forma regular software de alta calidad. Utilizando el PSP obtendrás datos que muestren la efectividad de tu trabajo e identifique tus puntos fuertes y tus debilidades. Esta herramienta es como las mediciones con cronómetro y de distancia que necesita hacerse uno mismo en relación con el ingreso en un equipo de atletismo y decidir en qué pruebas participar. Para tomar una decisión inteligente, necesitamos esas medidas para saber dónde destacamos y dónde necesitamos mejorar. Como un equipo de atletismo, la ingeniería del software tiene muchos especialistas, y los

ingenieros tienen una gran variedad de habilidades y talentos. Para tener una carrera satisfactoria y útil, necesitas conocer tus destrezas y habilidades, y sacar partido de tus talentos en el trabajo que haces. El PSP te ayudará a hacer esto.

LA UTILIZACIÓN DEL PSP

Utilizando el PSP, estarás practicando las habilidades y métodos que ingenieros del software profesionales han desarrollado durante muchos años de pruebas y errores. Basándote en las experiencias de tus predecesores, aprenderás más rápidamente y evitarás repetir sus errores. La esencia de ser un profesional es entender lo que otros han hecho antes que tú y construir sobre su experiencia.

CÓMO SE BENEFICIARÁN LOS ESTUDIANTES

Aunque el PSP se introduce actualmente en programas de ingeniería del software para licenciados, sus principios pueden ser aprendidos y practicados por estudiantes de los primeros cursos. Este libro está diseñado para introducir los métodos del PSP en pasos graduales de la misma forma que haces tus otros trabajos del curso. Al terminar de leer cada capítulo, *haz* los ejercicios. Estos te muestran cómo gestionar tu tiempo, cómo planificar y controlar tu trabajo y cómo producir regularmente programas de alta calidad.

Puesto que lleva su tiempo desarrollar las habilidades y hábitos de forma efectiva, deberías practicar los métodos del PSP en cada ejercicio de software. Si haces esto, habrás aprendido, practicado y perfeccionado estas habilidades antes de que las necesites en tu trabajo de ingeniero de software.

CÓMO PUEDEN UTILIZAR ESTE LIBRO LOS INGENIEROS QUE TRABAJAN

Los ingenieros de software también pueden utilizar este libro para aprender los fundamentos del PSP. Te sugiero que trabajes los ejercicios desde el principio del libro hasta el final, utilizándolos como guías para mejorar la forma de hacer tu trabajo cotidiano. Practica cada ejercicio hasta que lo hayas aprendido, entonces lee el siguiente capítulo e incorpora sus métodos. De nuevo, practica tanto los métodos nuevos como los aprendidos antes de avanzar al siguiente paso. La clave es dedicarle tiempo a dominar un método antes de seguir avanzando.

Con algo de dedicación y disciplina no tendrás problema en dominar esta materia. El éxito es más probable, sin embargo, si haces este trabajo

en una clase o con un grupo de colaboradores con quien puedas intercambiar experiencias y compartir ideas. En cualquier caso, planifica dedicar entre una o dos horas cada semana a estudiar el libro de texto, registrar y analizar tus datos del PSP, y adaptar los métodos a tu trabajo. Aunque el tiempo que necesitas para aprender el PSP dependerá de tus hábitos actuales y de tus prácticas, una vez que hayas terminado esta materia, tendrás una base sólida para continuar desarrollándote profesionalmente. Observa, sin embargo, que la clave para aprender el PSP es examinar y pensar sobre los datos de tu trabajo y lo que estos datos te indican sobre tu rendimiento personal.

ALGUNAS SUGERENCIAS PARA LOS PROFESORES

Este libro está diseñado como un texto de apoyo para los cursos tradicionales de informática (CS) o ingeniería del software de duración bisemestral¹. Solo presupone una formación preuniversitaria. El libro presenta el PSP en una serie de pasos que los estudiantes pueden utilizar en sus trabajos programados durante el curso. Los ejercicios de los 10 primeros capítulos son bastante generales y pueden utilizarse con o sin trabajos de programación. Los ejercicios de los diez últimos capítulos están diseñados para realizarlos con pequeños programas, de seis a ocho o más.

Aunque algunos estudiantes aprenden a programar en la universidad, ahora muchos aprenden nociones básicas de programación en institutos de enseñanza secundaria. Este material está diseñado tanto para utilizarse en los primeros cursos de programación como en cursos más avanzados. Si los estudiantes ya saben cómo programar o están aprendiendo, deberían entender fácilmente el material y encontrarlo inmediatamente útil.

Este libro presenta solo una introducción al PSP, pero no lo describe por completo. El libro no trata, por ejemplo, las técnicas estadísticas necesarias para hacer estimaciones con exactitud o análisis de datos. Tampoco trata los métodos para dimensionar el PSP a grandes proyectos, o la definición de procesos y la mejora de técnicas utilizadas al aplicar el PSP a tareas distintas de las de codificar pequeños programas. El PSP completo debería enseñarse posteriormente a los estudiantes dentro del programa educativo².

¹ Tal y como se organizan las asignaturas en la Universidad española, el autor se refiere a una asignatura anual, organizada en dos cuatrimestres, en vez de dos semestres (N. del T.).

² El PSP y el curso del PSP están descritos con más detalle en mi libro de texto *A Discipline for Software Engineering* (Reading, MA: Addison-Wesley, 1995). El libro de texto lleva materiales que incluyen una guía para el profesor y un disquete con transparencias para clase, así como ejercicios.

Conforme los estudiantes son conducidos a través del material de este libro y completan sus ejercicios, aprenderán a controlar y monitorizar su trabajo, gestionar su tiempo y hacer planes. En el segundo semestre, aprenderán sobre la calidad de los programas, las formas de hacer revisiones y a utilizar varias medidas de calidad y métodos de gestión. Aprenderán sobre los defectos, sus causas y la responsabilidad personal del ingeniero en la calidad de los productos que hacen. Al final del segundo semestre del curso, los estudiantes habrán aprendido los principios básicos del PSP. A partir de estos principios básicos y aumentando la experiencia con estos métodos, se debería requerir a los estudiantes que en cursos posteriores continuasen utilizando el PSP.

UNA ESTRATEGIA DE ENSEÑANZA

Puesto que este libro pretende ser utilizado en combinación con un curso bisemestral de ingeniería del software o de introducción a la informática, el material está dividido en un primer semestre que trata sobre la gestión del tiempo (10 capítulos) y un segundo semestre que trata sobre la calidad. La enseñanza de este material supone unas seis horas de clase en los dos semestres. Puesto que los estudiantes utilizan los métodos del PSP al mismo tiempo que hacen otros trabajos del curso, este material no añade una carga de trabajo significativa a los estudiantes. El tiempo adicional que dedican los estudiantes a aprender estos métodos se compensa por la eficiencia que consiguen.

Conforme se avanza en el libro, se deben proponer los ejercicios de cada capítulo a los estudiantes. La experiencia indica que es mejor hacer los 10 primeros capítulos en las primeras semanas del primer semestre. Los estudiantes tienen entonces el resto del semestre para practicar los métodos introducidos. En el segundo semestre se debería seguir la misma estrategia, introducir los temas del PSP en las primeras semanas del semestre y posteriormente utilizar estos métodos durante el resto del semestre.

Es de gran importancia presentar este material como una parte integral del curso. Explicando que estos son métodos fundamentales de ingeniería del software, que los estudiantes deben aprender y practicar para completar de modo satisfactorio el curso. Cuando se proponga un ejercicio, se debe explicar que la evaluación de los estudiantes dependerá tanto de la calidad de su trabajo como de lo bien que apliquen los métodos del PSP. Deben hacer cada ejercicio del PSP y continuar utilizándolos después de esta primera introducción. La estrategia del curso, sugiere la impartición de los contenidos y los ejercicios incluidos en el libro y los materiales de apoyo descritos al final del mismo.

PREPARACIÓN DEL PROFESOR

Para la impartición de este curso, será útil que el profesor haya aplicado él mismo los métodos. Podría utilizar, por ejemplo, los métodos de planificación y gestión del tiempo para preparar las clases o evaluar el trabajo de casa. Después de utilizar personalmente el PSP, apreciará mucho mejor la disciplina personal requerida. Este conocimiento le ayudará a explicar el PSP a los estudiantes y no solo a guiarlos en su utilización. Cuando vean que el profesor ha utilizado el PSP, es más probable que ellos mismos lo utilicen.

AGRADECIMIENTOS

Al escribir este libro estoy especialmente agradecido a la Facultad de Informática de la Universidad Aeronáutica de Embry-Riddle. Ellos me animaron a desarrollar este libro y me revisaron amablemente el manuscrito. Puesto que habían impartido varios cursos y utilizado ellos mismos los métodos del PSP, hicieron muchas sugerencias útiles. Por su apoyo y ánimo, les doy particularmente las gracias a los profesores Tom Hilburn, Iraj Hirmanpour, Aboalfazl Salimi, Davie Srachet, y Massood Towhidnejad. También doy las gracias a sus tres estudiantes, Ben Bishop, Guillermo José Hernández y Richard Rickert, por compartir sus datos del curso y las experiencias de clase conmigo. Además, agradezco a Ben Bishop, Andrew Henderson y Michael Patrick su amable descripción de sus experiencias con este material en el prólogo de los estudiantes.

Varios amigos y compañeros del SEI y de otros centros han revisado amablemente el manuscrito y me han indicado sugerencias y comentarios útiles. Estoy agradecido a Steve Burke, Howie Dow, John Eikenberry, Andy Huber, Julia Mullaney, Glenn Rosander, Marie Silverthorn, y Bob Stoddard. Mi secretaria Marlene MacDonald ha sido de gran ayuda en la lectura y comentario de las partes de este manuscrito, y en la distribución y revisión del mismo. Doy las gracias a Peter Gordon, Helen Goldstein, y a la dirección de Addison-Wesley por su apoyo en la impresión de este libro.

Finalmente, he sido bendecido por una esposa maravillosa. Bárbara me ha proporcionado ayuda y estímulo como en los otros libros. Como en las otras obras, ha leído el manuscrito final antes de enviarlo a su publicación. A pesar de toda su ayuda y apoyo, estoy seguro de que ni Bárbara ni mis muchos colaboradores habrán encontrado todas mis equivocaciones y errores. Los errores que permanecen son completamente míos.

Watts S. Humphrey

Sarasota, Florida



CAPÍTULO 1

El trabajo del ingeniero del software

Además de describir el trabajo del ingeniero de software y algunas de sus actividades principales, este capítulo también da una visión de la estrategia de este libro para ayudarte a desarrollar y mejorar tu formación en ingeniería del software. El capítulo termina con un ejercicio que consiste en especificar las tareas más importantes que realizarás a lo largo del tiempo que trabajes con este libro.

1.1

¿QUÉ ES LA INGENIERÍA DEL SOFTWARE?

El trabajo de un ingeniero del software es entregar productos software de alta calidad a unos costes establecidos y en un plazo determinado. Hay así, tres aspectos que hacen efectivo un trabajo de ingeniero del software: producir productos de calidad, hacer el trabajo a los costes esperados y completar el trabajo de acuerdo con la planificación establecida. Después de años de dolorosas experiencias, muchos ingenieros de software han aprendido que para hacer un trabajo efectivo necesitan:

1. Planificar su trabajo.
2. Hacer su trabajo de acuerdo con el plan.
3. Esforzarse en producir productos de máxima calidad.

El principal objetivo de este libro es mostrarte cómo hacerlo.

1.2**¿POR QUÉ ES IMPORTANTE UNA BUENA INGENIERÍA?**

Hasta ahora, son pocas las organizaciones de software que han satisfecho de forma fiable sus compromisos de costes y planificación. Este pobre record no solamente da a la ingeniería del software una mala imagen, también causa serios problemas en los negocios. Hay muchos ejemplos de negocios fracasados, disputas de contratos, litigios y molestias a los clientes. Entre ellos: el multimillonario sistema de control de tráfico aéreo de la **FAA** que duplicó su coste y sufrió repetidos retrasos de entrega debido a problemas de software, y Ashton Tate, una gran empresa de software, quebró debido a la pobre calidad de sus productos software. Los defectos del software han causado pérdidas humanas [Leveson].

El software de los ordenadores es actualmente crítico para muchos negocios. Funciona en la mayoría de las modernas industrias, maneja diariamente las transferencias internacionales de billones de dólares, y es un elemento clave en los nuevos productos y servicios que utilizamos. Puesto que la importancia del software en los negocios aumenta, la eficacia de los grupos de ingeniería del software es cada vez más importante. Por lo tanto, tu activo más importante como ingeniero será tu capacidad para hacer coincidir consistentemente tus compromisos con la calidad de los productos.

1.3**EL PROCESO SOFTWARE PERSONAL**

El Proceso Software Personal (**PSP**)SM fue diseñado para ayudar a los ingenieros del software a hacer bien su trabajo. Muestra cómo aplicar métodos avanzados de ingeniería a sus tareas diarias. Proporciona métodos detallados de planificación y estimación, muestra a los ingenieros cómo controlar su rendimiento frente a estos planes y explica cómo los procesos definidos guían su trabajo.

El Proceso Software Personal completo puede ser enseñado en un curso universitario de 15 clases (15 lecciones de una hora), donde los estudiantes practican los métodos del PSP mientras completan 10 ejercicios de programación y 5 ejercicios de análisis. Esto les ayuda a entender cómo el PSP trabaja para ellos. El PSP se ha impartido en un gran número de universidades y está siendo introducido en muchas organizaciones industriales. Los datos reunidos de los cursos impartidos muestran que el PSP es efectivo a la hora de mejorar el rendimiento en la planificación de los ingenieros y la calidad de sus productos.

SM Personal Software Process y PSP son marcas registradas de la Universidad de Carnegie Mellon.

El **PSP** también es efectivo en la industria del software. En un caso, antes de aprender el **PSP**, un grupo de ingenieros necesitó, en promedio, unas cinco veces más de lo que habían estimado para desarrollar tres componentes de un sistema software. Después del entrenamiento con el **PSP**, los mismos ingenieros terminaron los seis siguientes componentes del mismo producto en un 10,4% menos del tiempo que habían planificado. Cuando midieron los defectos encontrados por los clientes, la calidad de los componentes terminados después del curso del **PSP** fue cinco veces mejor que la calidad de los primeros componentes programados.

Cada capítulo de este libro, introduce un método del **PSP** que tú deberías aplicar posteriormente en tu trabajo. Utilizándolo, verás cómo funciona el método y adquieres práctica en su aplicación.

Supone un considerable esfuerzo y perseverancia aplicar el método del **PSP** de una forma consistente, pero esa, es la única forma de aprenderlo. Las clases del curso y el libro de texto son importantes, pero el principal medio de aprendizaje son los datos que reúnas en tu propio trabajo conforme completas cada ejercicio del **PSP**. Es importante tener estos datos para que puedas ver posteriormente cómo has mejorado tu rendimiento con el **PSP**.

1.4

LA DISCIPLINA DEL TRABAJO DE ALTA CALIDAD

La disciplina se define como una actividad o ejercicio que desarrolla o mejora habilidades. Contrariamente a la visión popular de la disciplina como una limitación onerosa, es un marco de trabajo para aprender y mejorar personalmente. La disciplina del **PSP** proporciona un marco de trabajo estructurado para desarrollar las habilidades personales y los métodos que necesitarás como ingeniero del software. La cuestión no es si tú necesitas habilidades personales sino cuánto tiempo necesitas para desarrollarlas y cómo las utilizas de forma consistente. La disciplina de **PSP** acelerará tu aprendizaje.

Los profesionales en muchos otros campos aprenden y practican las habilidades y los métodos de sus profesiones durante su educación formal. Los químicos aprenden a manejar los equipos del laboratorio y a hacer análisis precisos. Antes de ejercer la cirugía, los médicos internos observan y practican bajo la supervisión de cirujanos expertos. Ninguno de ellos pensaría en tener que aprender cuestiones de higiene, infecciones o procedimientos de quirófano en el trabajo. Los pilotos de líneas aéreas están muy cualificados antes de sus primeros vuelos comerciales, y los músicos dedican años a ensayar antes de su debut. Así, en otros campos, los profesionales demuestran su competencia básica antes de que se les permita desarrollar el trabajo más simple, mientras que en el software, los

ingenieros sin entrenamiento en el PSP deben aprender las habilidades que necesitan en el trabajo. Esto no solo es costoso y consume tiempo, sino que aumenta el riesgo.

1.5

LA IMPORTANCIA DEL TRABAJO DE ALTA CALIDAD

Como ingeniero del software en prácticas, probablemente desarrollarás partes de grandes productos o sistemas. No importa cómo sean de pequeñas con respecto al tamaño total del producto o su escasa importancia, cualquier defecto en ellas podría potencialmente dañar todo el sistema. La calidad de un sistema software no solamente está gobernada por la calidad de sus partes, sino que cualquier error trivial en los programas de soporte puede tener efectos devastadores.

Los sistemas informáticos modernos pueden ejecutar millones de instrucciones por segundo. Así, un defecto poco probable que pueda suceder solamente una vez de entre un billón puede ocurrir varias veces en un día. Con el software, las condiciones inusuales se presentan todas las veces, condiciones que parecen imposibles ocurren en poco tiempo. Los defectos en los elementos más pequeños de un gran sistema pueden causar serios problemas de forma impredecible y ocasional. Si cometes un error trivial que deja un defecto en el producto, el resultado puede causar un gran inconveniente o incluso un daño físico al usuario.

Para producir sistemas software de calidad, cada ingeniero debe aprender a hacer el trabajo con calidad. Si aprendes de forma consistente a producir programas de gran calidad, tú y tus productos seréis muy valorados por tus jefes y por tus clientes.

1.6

¿CÓMO MEJORAR LA CALIDAD DE TU TRABAJO?

Cuando estaba en la Marina de los Estados Unidos tuve que aprender a disparar un fusil. El entrenamiento era con fusiles y platos de barro. Mis puntuaciones eran pésimas y no mejoraban, aunque practicase. Después de observarme, el instructor me sugirió que intentase disparar con la mano izquierda. Siendo diestro, al principio encontré eso ilógico, después de unos pocos disparos de prueba, yo mejoré mis puntuaciones.

Hay varias cuestiones en este ejemplo. Primero, las medidas son necesarias para diagnosticar un problema. Sabiendo cuántos fallos y aciertos cometía, el instructor y yo podíamos ver fácilmente qué cosas tenía que hacer de forma diferente. A continuación, teníamos que utilizar esas medidas en algún tipo de análisis objetivo. Observándome, el instructor podía analizar el proceso que yo seguía para disparar el fusil, los pasos

que hacía eran: cargar, posicionarme, localizar, apuntar y disparar. El objetivo del instructor era descubrir qué pasos eran la fuente de mi problema. Él rápidamente se centró en el objetivo y sugirió que yo tenía que hacer cambios.

Finalmente, y más importante, vino el cambio mismo. El proceso de mejora es difícil porque las personas son reacias a intentar cosas nuevas. Sus hábitos parecen tan naturales que pueden pensar que el cambio no les ayudará. Yo siempre había sido diestro y nunca pensé disparar con la izquierda. Una vez que hice el cambio sugerido, mis puntuaciones mejoraron.

Definir medidas no es siempre fácil, pero al menos no es imposible. Este libro define varias medidas para el proceso software. Una vez que has definido las medidas para tu trabajo, debes reunir y analizar los datos. Si necesitas mejorar, a continuación analizas el proceso para ver dónde tienes que hacer cambios. Finalmente, para mejorar, debes cambiar lo que haces normalmente.

Si no hubiese cambiado mi proceso, podía haber estado intentándolo durante años sin mejorar como tirador. Las medidas, por si solas no producirán mejora aunque se repitan frecuentemente una y otra vez. Si continúas trabajando como siempre lo has hecho, continuarás produciendo los mismos resultados de siempre.

1.7

EL PROCESO DE MEJORA

Los pasos necesarios para cambiar la forma de tu trabajo son los mismos que los pasos que seguí para aprender a disparar. No son complicados. El proceso de mejora se muestra en la Figura 1.1.

- Definir el objetivo de calidad. Obviamente, mi objetivo era darle a la diana el mayor número de veces posible, el 100% era el objetivo final.
- Medir la calidad del producto. El instructor y yo podíamos ver que mis puntuaciones eran malas y algo había que hacer.
- Entender el proceso. El instructor observó qué hacía yo para ver qué es lo que debía de cambiar.
- Ajustar el proceso. Él sugirió que tenía que disparar con la mano izquierda.
- Utilizar el proceso ajustado. A continuación, disparé varias veces, pero esta vez con la izquierda.
- Medir los resultados. Contamos el número de aciertos y fallos.

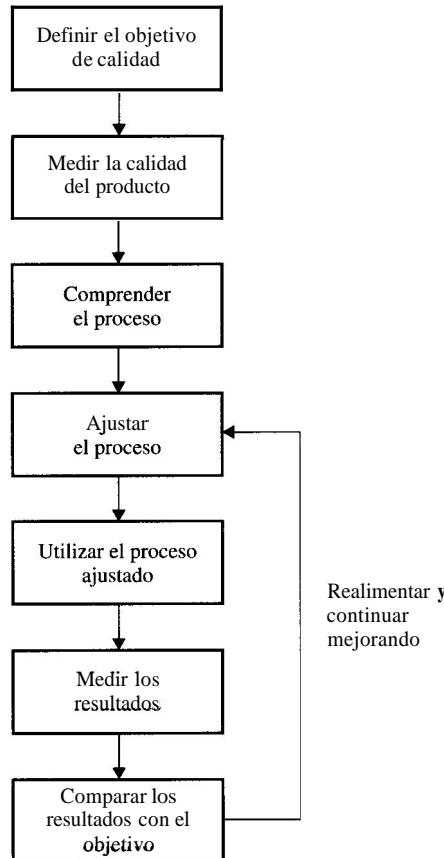


Figura 1.1 El proceso de mejora.

- Comparamos los resultados con el objetivo. A partir de esos datos, pudimos ver que mis puntuaciones mejoraron bastante.
- Realimentar y mejorar continuamente. Puesto que aprender a disparar un fusil era sencillo, el proceso de mejora continua no fue necesario.

Con procesos más complejos, generalmente se requieren varios ciclos. Con un proceso sofisticado como el desarrollo del software, los ciclos de mejora no deberían terminar nunca.

1.8

EL OBJETIVO DE ESTE LIBRO

Este libro introduce algunos métodos clave de ingeniería del software, uno por uno. Estos métodos son todos importantes, pero deben enseñarse

en algún orden. Esto es un poco como enseñar a alguien a aprender a nadar. Puedes hablar sobre la respiración, los movimientos de piernas, la flotación y los movimientos de brazos como actividades individuales, pero tienes que hacerlas todas a la vez para nadar. Generalmente enseñarás primero la respiración, pero el orden en el cual expliques el resto de cosas no lo hace muy diferente. Debes, sin embargo, cubrir todos los aspectos. En cualquier caso, hasta que no los hagas todos juntos, no puedes nadar. Con este libro, somos más afortunados. Muchos de estos métodos pueden ser utilizados independientemente y te ayudarán de alguna forma. Dichos métodos, sin embargo, se refuerzan unos a otros. Por lo tanto, hasta que no los utilices todos, no conseguirás un beneficio **global** del proceso disciplinado de la ingeniería del software. De la misma forma que debes primero aprender a respirar en el agua antes de poder nadar, en la ingeniería del software disciplinada debes comenzar por reunir datos.

El objetivo de este libro es introducirte en los pasos del proceso de mejora que se muestran en la Figura 1.1. Puesto que el orden no es crítico, he decidido presentar este material en dos partes. Los 10 primeros capítulos del libro tratan de la planificación y gestión del tiempo: procesos de planificación. Los 10 últimos capítulos tratan de la calidad del producto: proceso de gestión de defectos. Es importante tratar primero el proceso de planificación porque proporciona una base sólida para el proceso de gestión de defectos.

RESUMEN

Los ingenieros del software deberían planificar su trabajo, y actuar de acuerdo con dicho plan. El resultado serán productos de alta calidad ajustados a un presupuesto y unos plazos. El Proceso Software Personal (PSP) es un marco de trabajo diseñado para enseñar a **los** ingenieros del software a hacer mejor su trabajo. Muestra cómo estimar y planificar tu trabajo, cómo controlar tu rendimiento frente a esos planes y cómo mejorar la calidad de **los** programas que produces.

Los métodos de calidad lleva tiempo aprenderlos y practicarlos, pero te ayudarán durante tu carrera como ingeniero. Para mejorar consistentemente la calidad de tu trabajo, debes establecer objetivos, medir la calidad del producto, entender el proceso, cambiar y reutilizar el proceso, medir y analizar los resultados y finalmente realimentar y mejorar continuamente.

La estrategia de este libro es mostrarte cómo organizar y planificar tu trabajo, y cómo gestionar la calidad de los programas que produces. El primer paso para entender tu proceso es identificar las tareas que haces, estimar cuánto tiempo necesitará cada tarea, y medir el tiempo gastado.

EJERCICIO 1

El primer paso para entender tu proceso es identificar las tareas que haces. Por ejemplo, como estudiante, asistirás a clase, escribirás programas, leerás libros de texto y harás varios trabajos en casa. En algún momento, necesitarás estudiar para los exámenes. Parte del trabajo de casa consistirá en escribir programas. Una forma de describir estas tareas podría ser la que se muestra en la Tabla 1.1. Aquí, el Estudiante X espera dedicar unos 1200 minutos a sus tareas semanales y otros 300 minutos una vez al semestre a estudiar exámenes. Esto da un total de 20 horas cada semana y otras 5 horas para estudiar los exámenes.

Tarea	Frecuencia	Tiempo (minutos)
Asistir a clase	L, M, V	180/semana
Leer libros de texto	Semanal	180/semana
Trabajo en casa	Semanal	420/semana
Escribir programas	Semanal	420/semana
Preparar exámenes	Una vez al semestre	300/semestre
Revisar apuntes	Durante el trabajo de casa, estudiar	Incluido en otros tiempos

En este ejercicio, debes definir tus principales actividades para este curso y escribirlas en un formato como el indicado en la Tabla 1.1. Después de hacer esta lista, estima la frecuencia de dichas tareas y cuánto tiempo le dedicarás a cada una. Para todas las tareas semanales, estima el tiempo que utilizarás cada semana, y para las tareas mensuales o semestrales, estima los tiempos mensuales o semestrales. Este ejercicio no requiere que midas esos tiempos. Una estimación es suficiente. Haz y guarda una copia de esta lista antes de hacer tu trabajo de casa. Asegúrate de poner tu nombre y la fecha.

REFERENCIA

[Leveson] Leveson, Nancy G. *Safeware, System Safety and Computers*. Reading, MA: Addison-Wesley, 1995.

CAPITULO 2

La gestión del tiempo

Este capítulo muestra cómo gestionar tu tiempo y por qué es importante hacerlo. También describe cómo y por qué utilizar un cuaderno de ingeniería. Como ejercicio del capítulo harás un cuaderno de notas de ingeniería para utilizarlo en tu trabajo.

2.1 LA LÓGICA DE LA GESTIÓN DEL TIEMPO

Los fundamentos para gestionar el tiempo son:

Probablemente harás esta semana lo mismo que hiciste la semana pasada. En general, la forma en que utilizaste tu tiempo la Última semana te proporcionará una aproximación bastante buena a la forma en la que gastarás tu tiempo en futuras semanas. Hay, sin embargo, muchas excepciones. Durante la semana del examen, por ejemplo, no puedes asistir al mismo número clases y probablemente dedicarás más tiempo a estudiar y menos a hacer trabajos en casa.

Para hacer un plan realista, tienes que controlar tu forma de gastar tu tiempo. Aunque recuerdes cómo gastaste tu tiempo la Última semana, te sorprenderías de tus datos reales. Las personas recuerdan algunas cosas y olvidan otras. Por ejemplo, el tiempo que utilizaste en hacer trabajo en casa es probablemente mucho menor de lo que estimaste, mientras que el tiempo de comer o de relajarte con los amigos, es con frecuencia, muy superior al esperado. Nuestra memoria tiende a minimizar el tiempo que de-

dicamos a cosas que parecen que transcurren rápidamente, porque nos agrada hacer dichas cosas. Por el contrario, en las actividades lentes, aburridas o difíciles parece que se dedica más tiempo del que realmente se consume. Por lo tanto, para saber cómo utilizar tu tiempo, necesitas tener registros exactos del mismo.

Para comprobar la exactitud de tus estimaciones de tiempo y planes, debes documentarlas y posteriormente compararlas con la que realmente haces. Mientras esto no es un problema serio en las universidades, es de importancia crítica para el trabajo de los ingenieros. La planificación es una habilidad que pocas personas han aprendido. Hay, sin embargo, métodos de planificación conocidos que se pueden aprender y practicar. El primer paso para aprender a hacer buenos planes, es hacer planes. Así que, escribe tu plan para que posteriormente tengas algo con lo que puedas comparar tus datos actuales.

Para hacer más precisos tus planes, determina las equivocaciones de los planes anteriores, y qué podrías haber hecho para mejorar. Cuando hagas el trabajo planificado, registra el tiempo que utilizas. **Esos** datos del tiempo serán útiles si se anotan con un poco de detalle. Por ejemplo, cuando estés haciendo el trabajo del curso, registra por separado el tiempo que dedicas a asistir a clase, leer libros de texto, escribir programas y estudiar para los exámenes. Cuando codifiques grandes programas, de igual forma encontrarás útil registrar los tiempos para las distintas partes del trabajo: diseño del programa, escritura del código, compilación y pruebas. Aunque dicho grado de detalle no es necesario para programas muy cortos, puede ser útil cuando trabajes en proyectos que necesiten varias horas o más.

Cuando tengas la copia documentada de tu plan y hayas registrado a qué has dedicado tu tiempo, puedes comparar fácilmente los resultados reales con el plan original. Verás donde estaba equivocado el plan y como tu proceso de planificación puede ser mejorado. La clave para planificar con exactitud es hacer planes consistentes y compararlos con los resultados reales posteriores. Entonces verás como puedes hacer planes mejores.

Para gestionar tu tiempo, planifica tu tiempo y sigue el plan. Determinar qué podrías hacer para producir mejores planes es la parte más fácil. Llevarlo a cabo es lo realmente difícil. El mundo está lleno de resoluciones que nunca se cumplen, como seguir una dieta o dejar de fumar.

Al principio, cumplir un plan es probablemente difícil. Hay muchas razones posibles, pero la más común es que el plan no era muy bueno. Hasta que no intentes seguirlo, probablemente no sabrás porque. Trabajando con el plan, consigues el primero de dos beneficios: saber dónde estaba equivocado el plan, lo cual te ayudará a mejorarlo en el próximo proyecto.

El segundo beneficio de trabajar con el plan es que harás el trabajo de la forma que lo has planificado. Puede que esto no parezca muy importan-

te, pero lo es. Muchos de los problemas en la ingeniería del software son causados por atajos irreflexivos, descuidos y distracciones en los detalles. En muchos casos, los propios métodos eran conocidos y especificados pero no se seguían. Aprender a establecer planes útiles es importante, pero aprender a seguir dichos planes es absolutamente crucial.

Otro beneficio más sutil de trabajar de acuerdo a un plan es que cambias tu comportamiento actual. Con un plan, es menos probable que deseches tiempo en decidir qué harás después. El plan también te ayuda a centrarte en lo que estás haciendo. Es menos probable que te distraigas y es más fácil ser eficiente.

2.2

COMPRENDE CÓMO UTILIZAS EL TIEMPO

Para practicar la gestión del tiempo, el primer paso es entender cómo utilizas el tiempo ahora. Esto se hace en varios pasos:

Clasifica tus principales actividades. Cuando comiences a controlar el tiempo, probablemente encontrarás que gran parte del mismo lo dedicas a relativamente pocas actividades. Esto es normal. Para hacer algo, debemos centrarnos en pocas cosas que sean muy importantes. Si distribuyes tu tiempo entre muchas cosas, será difícil encontrarle sentido a los datos. De tres a cinco categorías deberán ser suficientes para controlar el tiempo durante el curso. Si posteriormente necesitas un mayor grado de detalle, divide las categorías más generales en subcategorías.

Registra el tiempo dedicado a cada una de las actividades principales. Se necesita bastante disciplina personal para registrar el tiempo de forma consistente. Toma un registro exacto, registra el tiempo de inicio y fin de cada actividad principal. Al principio lo olvidarás con frecuencia, pero después de cierta práctica será natural en ti. El Capítulo 3 describe el registro del tiempo con más detalle.

Registra el tiempo de forma normalizada. Normalizar los registros de tiempo es necesario porque el volumen de datos aumentará rápidamente. Si no registras y almacenas cuidadosamente estos datos, se perderán o estarán desorganizados. Los datos confundidos o desordenados son difíciles de encontrar o interpretar. Si no intentas tratar los datos de forma adecuada, puede que no los reúnas bien. El Capítulo 3 describe una tabla normalizada de registro de tiempos, utilizada en el PSP para reunir datos.

Guarda los datos de tiempo en un lugar adecuado. Puesto que necesitarás guardar los registros de tiempo con los trabajos del curso, guárdalos en un lugar adecuado. Esta es una de las principales utilidades del cuaderno de ingeniería.

2.3**EL CUADERNO DE INGENIERÍA**

En este curso, utilizarás un cuaderno de ingeniería para controlar el tiempo. Lo utilizarás también para otras cosas, tales como, guardar los ejercicios, controlar compromisos, tomar notas de clase y como un cuaderno de trabajo para anotar ideas de diseño y cálculos.

Como un profesional del software, le darás múltiples usos al cuaderno de ingeniería tales como: registrar los tiempos, guardar los cálculos y tomar notas de diseño. Podrás utilizarlo como una evidencia de lo que haces en la práctica de la ingeniería, evidencia importante para la defensa de tu empresa, si es que tienes que defender la responsabilidad legal de un producto. Cuando las partes perjudicadas demandan a la compañía, su principal objetivo es demostrar que los suministradores fueron negligentes. Para la compañía, la mejor defensa es la evidencia de que los ingenieros siguieron las prácticas de ingeniería. Por esta razón tener un cuaderno de ingeniería es un buen hábito.

Una utilización adicional del cuaderno de ingeniería es la protección de los activos intelectuales de los empleados, por ejemplo, registrando ideas que se puedan patentar. Una vez, en una reunión de diseño, entre mis colegas y yo ideamos algo que se podía considerar como una idea a patentar. Escribimos la idea en mi cuaderno de ingeniería y todos firmamos cada página. El asesor de patentes nos dijo que esto podría ser útil para establecer la fecha del invento. La compañía también nos dio a cada uno de nosotros un premio en metálico.

Aunque probablemente estas ideas no te interesen como estudiante, este curso trata sobre cómo aprender los métodos y establecer los hábitos que necesitarás en la práctica como ingeniero. Por ello, deberías disponer a partir de ahora de tu propio cuaderno de ingeniería y crearte el hábito de utilizarlo.

2.4**EL DISEÑO DEL CUADERNO**

El diseño particular del cuaderno no es clave, pero la práctica general en la industria es utilizar un cuaderno de gusanillo. Si numeras cada página, el diseño de gusanillo te permitirá tener las páginas en orden y un registro legal útil de tu trabajo. La desventaja, por supuesto, es que tendrás que registrar tus notas en orden cronológico y no podrás insertar o eliminar páginas fácilmente.

Una sugerencia para la portada de tu cuaderno de ingeniería se puede observar en la Tabla 2.1. En la parte superior, deberías etiquetar el cuaderno con un número de cuaderno. Después de haber guardado los cuadernos de ingeniería durante varios años, dispondrás de bastantes. La

Tabla 2.1 Ejemplo de portada del cuaderno de ingeniería.

		Cuaderno número: <u>1</u>
Cuaderno de ingeniería Nombre de la Empresa o Universidad		
Nombre del Ingeniero	<u>Jane Doe</u>	
Teléfono/correo electrónico	<u>id@db.xvz.edu</u>	
Fecha de apertura	<u>9/9/96</u>	Fecha de cierre

numeración de los cuadernos es conveniente para almacenarlos en orden cronológico. También, etiqueta cada cuaderno con tu nombre y número de teléfono o dirección de correo electrónico. Escribe la fecha de inicio de introducción de datos en el cuaderno, y cuando lo hayas terminado escribe la fecha del último registro.

Dentro del cuaderno, numera cada página, utiliza las dos primeras páginas para una breve tabla de contenidos. En los contenidos, escribe cualquier referencia especial para que puedas encontrarla posteriormente, por ejemplo: ejercicios del curso. Esto te evitará que tengas que buscar por to-

do el cuaderno. No es necesario registrar los contenidos por páginas si no esperas referenciarlos en el futuro.

2.5

EJEMPLOS DE CUADERNOS DE INGENIERÍA

Un ejemplo de la página de contenidos del cuaderno de ingeniería se muestra en la Tabla 2.2. Para materias que necesitarás en el futuro, escribe a la izquierda el número de la página del cuaderno con una breve des-

Tabla 2.2 Ejemplo de la página de contenidos del cuaderno de ingeniería.

cripción del tema. Por ejemplo, el estudiante registra en la página 3 todos los ejercicios de la asignatura de IP (Introducción a la Programación) para dos semanas. Los contenidos también muestran que **los** ejercicios se continuarán registrando en la página 11. Un ejemplo de la página 3 del cuaderno se muestra en la Tabla 2.3.

Los contenidos también muestran que entre el 9/9 y el 13/9, el estudiante tomó notas de clase en las páginas 4, 5, 6 y 7. Después, continuó

Tabla 2.3 Ejemplo de una página del cuaderno de ingeniería.

Fecha	3
919	Ejercicio IP, entrega 1619
	Hacer un cuaderno de ingeniería
	Referencia, página 206 del libro de texto
	Leer el texto de programación, capítulo 1
1119	Ejercicio IP, entrega 20/9
	Hacer ejercicios de programación, capítulo 1
1319	Ejercicio IP, entrega 23/9
	Leer el texto de programación del capítulo 2
	Revisar los ejercicios del capítulo 2 y preparar para el examen
1619	Ejercicio IP, entrega 2319
	Completar los registros de tiempos
	Utilizar la tabla de registros de tiempos
	Poner las páginas de registro al final del cuaderno de ingeniería
	Ver los ejemplos del capítulo 3 del libro de texto
	(los ejercicios continúan en la página 11)

tomando notas en la página 10. Siempre que tengas que saltar algunas páginas debido a otras anotaciones, es una buena idea escribir en la parte inferior de la página dónde continúa ese tema. Véase, por ejemplo, la Última línea de la Tabla 2.3.

RESUMEN

Para gestionar tu tiempo efectivamente, necesitas planificar tu tiempo y seguir el plan. Para hacer planes realistas, primero tendrás que controlar la forma de utilizar el tiempo. Debes documentar tus planes y compararlos con lo que haces realmente. Para mejorar la precisión de tu planificación, determina los errores de los planes anteriores y qué podrías hacer para mejorar el plan.

El primer paso para la planificación y gestión del tiempo es entender cómo utilizas tu tiempo realmente. Para hacer esto, necesitas clasificar tus actividades en categorías principales, entonces, de una forma normalizada, registra el tiempo dedicado a cada actividad. Para guardar estos datos en un lugar adecuado, utiliza el libro de ingeniería. En este curso utilizarás el libro de ingeniería para registrar los ejercicios, tomar notas y también como libro de trabajo para el diseño de ideas y la realización de cálculos.

EJERCICIO 2

Prepara un cuaderno de ingeniería para el trabajo que tengas que hacer en este curso. La portada debería incluir los puntos descritos en la Sección 2.4 y su tabla de contenidos debería incluir al menos una entrada para los ejercicios del curso o los compromisos de trabajo para la semana actual. Muestra el cuaderno al profesor en la siguiente clase. Después, el profesor podrá ocasionalmente pedirte el cuaderno de ingeniería para revisarlo. Observa que esto es un ejercicio opcional. No necesitas hacerlo a menos que te lo pida el profesor.

CAPITULO 3

El control del tiempo

Este capítulo describe un procedimiento y una forma de controlar y registrar la manera de gastar tu tiempo. También se dan ejemplos de los tipos de registros de tiempos a guardar. En el ejercicio harás y utilizarás una tabla de registro de tiempos.

3.1 ¿POR QUÉ CONTROLAR EL TIEMPO?

Como se describió en el Capítulo 1, la forma de mejorar la calidad de tu trabajo comienza por entender lo que haces realmente. Esto significa que tienes que conocer las tareas que haces, cómo las haces y los resultados que obtienes. El primer paso en este proceso es definir las tareas y averiguar cuánto tiempo gastas en cada una de ellas. Para hacer esto, debes medir tu tiempo real. Este capítulo describe cómo medir el tiempo y anotarlo en una tabla.

Mi experiencia escribiendo mi anterior libro me demuestra que esto puedes hacerlo. En la planificación para escribir *A Discipline for Software Engineering*, revisé mis datos de libros anteriores y encontré que había dedicado una media de 26 horas por semana a escribirlos. Esto me pareció mucho tiempo porque yo entonces trabajaba a tiempo completo y podía solamente dedicar las tardes y los fines de semana a escribir. Por ello, decidí planificar sobre 20 horas a la semana para cuando tuviese que es-

cribir un nuevo libro. A esta velocidad, yo podría acabar el manuscrito el siguiente mes de enero.

Después de estar trabajando en el nuevo libro durante un mes, encontré que el trabajo era mucho más difícil de lo que había previsto. Revisando mis datos, encontré que para cada capítulo le estaba dedicando más tiempo que el que había planificado. Y en vez de 20 horas por semana, le estaba dedicando realmente más de 30 horas. Aunque el trabajo era mayor de lo esperado, mis registros de tiempos me permitían hacer una estimación de que podría terminar el manuscrito cuatro meses después de lo que había planificado inicialmente. Y fue cuando realmente acabé.

Aunque no esperes escribir muchos libros, encontrarás mucho más fácil controlar la cantidad de tiempo que dedicas a cualquier proyecto que incrementar tu productividad. Cuando las personas dicen que trabajan de firme, quieren decir que están trabajando muchas horas. Si no sabes cómo utilizas realmente tu tiempo, no serás capaz de gestionar la forma de utilizarlo.

3.2

EL REGISTRO DE LOS DATOS DE TIEMPOS

Cuando registres el tiempo, recuerda que el objetivo es obtener datos de cómo trabajas realmente. La forma y el procedimiento utilizado para reunir los datos no es tan importante mientras los datos sean exactos y completos. Yo personalmente utilizo el método descrito en este capítulo y que he enseñado en los cursos a muchos estudiantes. También he enseñado esta aproximación a muchos ingenieros que han utilizado los métodos con un éxito considerable. Después de haber superado su resistencia natural a utilizar las tablas y procedimientos, muchos ingenieros encuentran los métodos sencillos y adecuados. A ti probablemente te sucederá lo mismo.

Conforme vayas trabajando con este libro, utiliza los métodos descritos en este capítulo para registrar el tiempo. Puedes preguntarte: ¿por qué debo utilizar esta tabla? ¿Por qué no diseño mi propia tabla? Las respuestas son:

- En una clase, no sería práctico tener a cada estudiante utilizando una tabla o método diferente.
- Sin una experiencia previa en la recogida de datos, sería difícil para ti idear una tabla y un procedimiento personal factibles.
- Después de acabar este curso, tendrás el conocimiento y la experiencia para modificar las tablas y procedimientos para adaptarlos a ti.

En otras palabras, utiliza los métodos descritos aquí. Así, después de haber intentado utilizar estos métodos en este curso, tienes libertad para cambiarlos o diseñar algo completamente nuevo. No importa el formulario que utilices, pero, si quieras gestionar tu tiempo debes controlarlo continuamente.

3.3 EL CONTROL DEL TIEMPO

Cuando las personas hablan generalmente sobre lo que hacen, a menudo utilizan las horas como medida. Pero esto no es muy útil. La razón es porque tú, realmente, no dedicas una hora completa a algo. Por ejemplo, en la Figura 3.1 mostramos un resumen de tiempos de 7 estudiantes tomados de un curso reciente en donde había que escribir algún ejercicio de codificación relativamente pequeño. Estos ejercicios requirieron una media de cuatro a seis horas cada uno, y los estudiantes tuvieron que controlar cuándo comenzaban y paraban su trabajo. Como puedes observar, el 90% de los períodos de trabajo de los 7 estudiantes fueron inferiores a una hora.

La cantidad típica de tiempo no interrumpido que los ingenieros dedican a sus tareas es generalmente inferior a una hora. Medir el trabajo en unidades de horas no proporcionará el detalle necesario para posteriormente planificar y gestionar tu trabajo. Es mucho más fácil controlar el tiempo en minutos que en fracciones de una hora. Considera, por ejemplo, los registros de tiempos que obtendrías si utilizases fracciones de hora. Las entradas serían números como 0,38 horas o 1,27 horas. Aunque las unidades de horas pueden ser más útiles en resúmenes mensuales o semanales, estas cantidades fraccionadas son difícil de calcular y complicado de interpretar. Por ejemplo, en vez de poner 0,38 horas, es más fácil de entender y de registrar 23 minutos. Una vez que has decidido controlar el tiempo, es más fácil controlarlo en minutos que en horas.

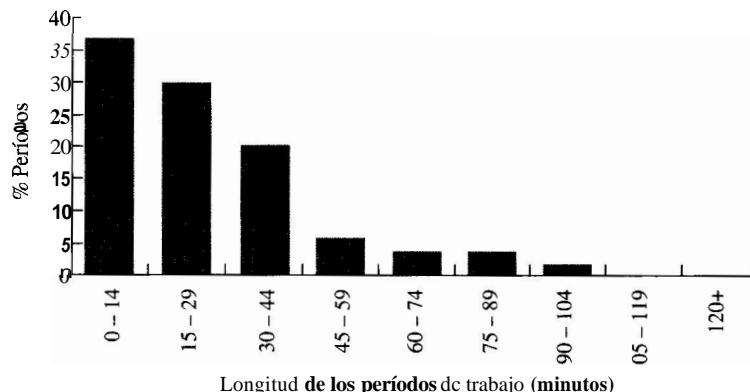


Figura 3.1 Períodos de trabajo del estudiante.

3.4

EL USO DE UNA TABLA DE REGISTRO DE TIEMPOS NORMALIZADA

La tabla utilizada para registrar el tiempo se muestra en la Tabla 3.1. La parte superior de la tabla, llamada cabecera tiene unos espacios para poner tu nombre, la fecha de comienzo, el nombre del profesor y el nombre o número de clase. Las columnas del cuerpo de la tabla son para registrar los datos de tiempo. Cada periodo de tiempo se introduce en una línea de la siguiente forma:

- Fecha. La fecha de realización de alguna actividad, como asistir a clase o escribir un programa.
- Comienzo. La hora de comienzo de la actividad.
- Fin. La hora a la que terminas de hacer la actividad.
- Interrupción. Cualquier pérdida de tiempo debida a interrupciones (véase la sección siguiente).
- Δ Tiempo. El tiempo dedicado a cada actividad en minutos, entre los tiempos de comienzo y fin menos el tiempo de interrupción.
- Actividad. Nombre descriptivo para la actividad.
- Comentarios. Una descripción más completa de lo que estás haciendo, el tipo de interrupción o cualquier cosa que podría ser útil cuando posteriormente analices los datos de tiempo. De nuevo, véase en la sección siguiente las interrupciones.
- C (Completado). Rellena esta columna cuando termines una tarea, como leer un capítulo del libro de texto o escribir un programa (véase 3.6).
- U (Unidades). El número de unidades de una tarea acabada (véase 3.6).

Un ejemplo de cómo llenar la tabla de registros de tiempos lo podemos ver en la Tabla 3.2. Aquí, la Estudiante Y escribe su nombre y la fecha en que comienza a llenar la tabla. También escribe el nombre del profesor y el nombre o número de curso. Cuando comenzó a guardar registros de tiempos el día 9 de septiembre, registró esa fecha en la parte superior del formulario y hace el primer registro en la parte superior izquierda. Su primera actividad fue asistir a la clase de IP a las 9:00 A.M. La clase duró hasta las 9:50, o sea un total de 50 minutos. La estudiante puso 9:00 en la columna de comienzo, 9:50 en la columna de fin y los 50 minutos de duración en la columna de Δ Tiempo. A la derecha, puso el nombre de la actividad en la columna denominada Actividad. Finalmente, en la columna Comentarios, anotó lo que estaba haciendo. Como la Estudiante Y continuó

haciendo actividades durante varios días, anotó **sus** otras actividades para el curso de IP en la tabla de registro de tiempos.

Tabla 3.1 Cuaderno de registro de tiempos.

Estudiante _____ Fecha _____

Profesor _____ Class _____

Registrar el tiempo de esta forma significa que hay que medirlo. Si no tienes un reloj, sería deseable conseguir uno.

3.5

LA GESTIÓN DE LAS INTERRUPCIONES

Un problema común con el control del tiempo son las interrupciones. Es sorprendente la frecuencia con que somos interrumpidos por llamadas telefónicas, personas que quieren charlar, molestias ocasionales o la necesidad de descansar. La forma de gestionar las interrupciones en el Cuaderno de Registro de Tiempos, consiste en anotarlas en la columna Tiempo de Interrupción. Como se observa en la Tabla 3.2, por ejemplo, la Estudiante Y comenzó su trabajo de casa a las 11:06 del día 10 y paró a las 12:19. Durante dicho período, tuvo dos interrupciones, una de 6 minutos y otra de 5. El tiempo neto que dedicó a su trabajo de casa fue de 73 minutos menos 11 minutos de interrupción, es decir un total de 62 minutos. La Estudiante Y no solamente anotó estos tiempos en el Cuaderno de Registro de Tiempos, sino que también describió brevemente las interrupciones en la columna de comentarios.

Una cosa que he encontrado útil, es utilizar el cronómetro para controlar las interrupciones. Cuando me interrumpen, pongo en marcha el cronómetro y cuando estoy preparado para reanudar el trabajo, lo paro. Antes de comenzar a trabajar de nuevo, anoto el tiempo de interrupción en el cuaderno y pongo a cero el cronómetro. Encuentro esto más adecuado que apuntar el tiempo de inicio de cada interrupción. Esto es también más exacto que estar adivinando. Un ingeniero que jugaba al ajedrez me dijo que encontró un reloj de ajedrez especialmente adecuado para controlar el tiempo de las interrupciones. La técnica que utilices no es tan importante mientras te sea útil.

Puesto que el tiempo de las interrupciones no es tiempo de trabajo productivo, debes controlar las interrupciones. Si la cantidad de este tiempo fuese constante, no tendrías que hacer mucho para gestionarlo. El tiempo de las interrupciones, sin embargo, es muy variable. Si no lo mides, deberías de añadir un número aleatorio en todos los datos de tiempos. Esto haría más difícil utilizar estos datos para planificar o gestionar tu tiempo.

Los datos de tiempo registrados pueden utilizarse para comprender con qué frecuencia se interrumpe tu trabajo. Las interrupciones no son solamente un despilfarro de tiempo, sino que rompen tu ritmo de pensamiento, llevándote a la ineficiencia y al error. Comprender como eres interrumpido te ayudará a mejorar la calidad y eficiencia de tu trabajo. Algunos ingenieros me han dicho que aprender a controlar el número y

duración de las interrupciones era uno de **los** beneficios más importantes del control de **su** tiempo.

Tabla 3.2 Ejemplo del cuaderno de registro de tiempos.

Estudiante _____ Estudiante Y _____ Fecha 9/9/96
Profesor _____ Sr. Z _____ Class CS1

3.6

EL CONTROL DE LAS TAREAS FINALIZADAS

Para controlar cómo gastas tu tiempo, necesitas controlar los resultados producidos. Para la asistencia a clases o reuniones, por ejemplo, un registro del tiempo sería adecuado. Cuando desarrollas programas, lees los capítulos de tu libro o escribes informes, necesitas saber cuánto trabajo has realizado. Podrías calcular la productividad de la tarea. Un ejemplo podría ser cuánto tiempo necesitas para leer un capítulo de un libro o escribir un programa. Con este conocimiento, podrás mejorar la planificación de futuros trabajos.

Las columnas C y U de la derecha del Cuaderno de Registro de Tiempos, significan Completado (C) y Unidades (U). Estas columnas te ayudan a identificar rápidamente el tiempo dedicado a las distintas tareas y lo que has hecho. Utilizarás esta información en posteriores capítulos de este libro, comenzando por el Capítulo 4.

Aquí, una unidad es una unidad de trabajo. Cuando has leído un capítulo, has completado una unidad de trabajo. Un programa acabado es otra unidad de trabajo. En el Capítulo 6, introducimos más medidas detalladas de unidades de trabajo, como las páginas de texto o las líneas de un programa fuente. Entonces, las 11 páginas de un capítulo serían 11 unidades de trabajo. Mientras puedas contar las unidades de cualquier forma, eso es útil, las unidades más detalladas como las líneas de código o las páginas de un texto son generalmente más útiles que las unidades grandes de capítulos o programas. Por ahora, sin embargo, utilizaremos unidades grandes y aplazaremos la discusión de la medida de unidades más detalladas hasta el Capítulo 6.

Para llenar la columna C, comprueba cuando has terminado una tarea. En la Tabla 3.2, por ejemplo, la Estudiante Y lee el libro desde las 6:25 a las 7:45 el 9 de septiembre. Durante este tiempo, completó la lectura de dos capítulos. Por ello puso una X en la columna C (Completado). Para anotar las unidades de trabajo acabadas, puso un 2 en la columna de U (Unidades). Posteriormente, cuando resuma su trabajo de la semana, rápidamente podría ver estas columnas para encontrar los elementos acabados. Podría también determinar cuántas unidades de trabajo ha realizado y cuánto tiempo ha dedicado. En la Tabla 3.2, por ejemplo, hay tres actividades de tipo Texto, pero solamente dos de ellas tienen rellena la columna C. En estas dos filas, la Estudiante Y dedicó un total de $80 + 28 = 108$ minutos a leer un total de $2 + 1 = 3$ capítulos del libro. Utilizando estos datos, por término medio, lee un capítulo del libro en 36 minutos.

Para tener unos registros de tiempos exactos, es importante completar las columnas C y U cada vez que acabes una tarea que tenga resultados medibles. Si olvidas hacerlo, puedes fácilmente encontrar la información, pero es mucho más fácil rellenarlo en el momento que aca-

bas la tarea. Las instrucciones para llenar el Cuaderno de Registros de Tiempos están repetidas en la Tabla 3.3 para facilitar la tarea de controlarlas antes de empezar a utilizarlas y vuelve a comprobarlas después, para estar seguro de que estás registrando los tiempos de forma adecuada.

3.7

¿CÓMO MANTENER LA TABLA DE TIEMPOS EN EL CUADERNO DE INGENIERÍA?

Para mantener los registros de tiempos, debes tener una copia del cuaderno de tiempos a mano durante este curso. El cuaderno de ingeniería resulta ser un lugar adecuado para guardar el Cuaderno de Registros de Tiempos. Yo registro mi tiempo en la parte posterior de mi cuaderno porque trabajo con frecuencia en diferentes proyectos al mismo tiempo. Puesto que comienzo y finalizo los proyectos todos a la vez, he encontrado más fácil guardarlos en el cuaderno de notas de ingeniería que tener diferentes hojas de tiempos a la vez.

El método que he encontrado más adecuado, consiste en poner el Cuaderno de Registro de Tiempos en la Última página del cuaderno y avanza páginas hacia el principio. Cuando completo una página paso a la siguiente. Hago esto hasta que la documentación de la parte delantera coincide con los Cuadernos de Tiempos de la parte trasera. Cuando sucede esto, comienzo un nuevo cuaderno.

Puede haber razones por las que quieras guardar separados los registros de tiempos de cada proyecto. Cuando hagas un solo proyecto cada vez, es más adecuado utilizar páginas separadas. También, en este curso, tu profesor puede querer que registres el tiempo en Cuadernos de Registros de Tiempos separados porque son más fáciles de entregar y de calificar. Mientras registres tu tiempo, la técnica que elijas no tiene la mayor importancia.

3.8

IDEAS PARA REGISTRAR TU TIEMPO

El control del tiempo es sencillo en teoría. Unos pocos trucos, sin embargo, pueden ayudarte a hacerlo de forma más consistente y precisa.

- *Lleva siempre contigo el cuaderno de notas.* Esto parece obvio, pero al principio lo olvidarás algunas veces. Cuando ocurra, anota la información de los tiempos en una hoja aparte y cópiala en el cuaderno tan pronto como puedas. Si utilizas el cuaderno para registrar otras cosas importantes como notas de clase, ejercicios y citas, es más probable que lo lleves siempre.

Table 3.3 Instrucciones para los cuadernos de registro de tiempos.

Propósito	Este formulario es para registrar el tiempo dedicado en este curso. Utiliza las páginas del final del cuaderno de ingeniería para hacer el Cuaderno de Registros de Tiempos.
Método	Registra todos los tiempos de este curso. Registra el tiempo en minutos. Sé tan preciso como puedas.
Cabecera	Anota lo siguiente: Tu nombre y fecha actual. El nombre del profesor y el nombre o número de curso. Asegúrate que tu nombre está en todas las copias del Cuaderno de Registro de Tiempos.
Fecha	Anota la fecha de inicio de la tabla.
Ejemplo	14/9/96
Comienzo	Anota la hora de inicio de la tarea.
Ejemplo	9:15
Fin	Anota la hora de finalización de la tarea.
Ejemplo	11:59
Tiempo interrupción	Registra el tiempo de interrupción de la tarea y la causa de la interrupción.
Ejemplo	5 + 3 + 22, descanso, teléfono, charla.
Δ Tiempo	Anota el tiempo que dedicas a la tarea, menos el tiempo de interrupción.
Ejemplo	Desde las 9:15 a las 11:59, menos 30 minutos o 134 minutos.
Actividad	Anota el nombre de la tarea o actividad que estas realizando.
Ejemplo	Exámenes.
Comentarios	Anota cualquier comentario pertinente que posteriormente te permita recordar cualquier circunstancia inusual observada con relación a la actividad.
Ejemplo	Preparar exámenes.
C (Completada)	Cuando una tarea se termina, rellena este campo.
Ejemplo	A las 7:45 del 9/9, acabaste de leer uno o más capítulos, rellena este campo.
U (Unidades)	Anota el número de las unidades de trabajo acabadas.
Ejemplo	Desde las 6:25 a las 7:45 del 9/9 leíste 2 capítulos, escribe un 2.
Importante	Registra todos los tiempos de este curso. Si olvidas registrar un tiempo, anota la mejor estimación. Si olvidas el Cuaderno de Registros de Tiempos, anota estos tiempos y cópialos en tu tabla tan pronto como puedas.

- *Cuando ocasionalmente olvides registrar la hora de comienzo, la hora de fin o la duración de la interrupción, haz una estimación tan pronto como lo recuerdes.* Esto no será tan riguroso como registrar la hora exacta, pero es lo mejor que puedes hacer. Será lo más parecido al tiempo real.
- *Puedes utilizar un cronómetro para controlar las interrupciones.* Puede parecer excesivamente preciso, pero es más sencillo registrar el tiempo de inicio y de finalización de cada interrupción.
- *Resume tu tiempo puntualmente.* Utilizarás el Resumen Semanal de Actividades para resumir semanalmente tu tiempo en este curso. Cómo y por qué haces esto se discute en el Capítulo 4.

Otra aproximación es registrar los datos de tiempos en un computador. He intentado esto y he observado que lleva más tiempo y era menos adecuado que tomar notas en el papel. Los ordenadores podrían ser útiles para este propósito, pero sería necesario una aplicación adecuada. Pero hasta que se desarrolle dicha aplicación, puedes elegir entre escribirla tú mismo, utilizar un sistema manual de registro de tiempos o no controlar tu tiempo.

RESUMEN

Este capítulo describe cómo controlar tu tiempo. Controla el tiempo en minutos utilizando el Cuaderno de Registro de Tiempos, que deberías guardar al final de tu cuaderno de notas de ingeniería. Lleva contigo el cuaderno cuando hagas las tareas de este curso. Si olvidas el cuaderno, registra el tiempo de la tarea en una hoja aparte y cópiala en el cuaderno tan pronto como puedas. Considera la utilización de un cronómetro para controlar las interrupciones.

EJERCICIO 3

Utiliza el Cuaderno de Registro de Tiempos para controlar el tiempo que dedicas a las distintas actividades de este curso. Como parte de este ejercicio, examina la lista de actividades que pusiste en el Ejercicio 1 y ajústalas si es necesario. Entrega la próxima semana una copia de tu registro de tiempos. Será obligatorio entregar una copia del Cuaderno de Registro de Tiempos cada semana desde este momento hasta el final del curso.

CAPITULO 4

Planificación de períodos y productos

Este capítulo describe la planificación de períodos y de productos, y muestra cómo relacionarlos con tu trabajo personal. Aprenderás a llenar un Resumen Semanal de Actividades para los datos del Cuaderno de Registro de Tiempos, que lleves anotados hasta ahora en el curso.

4.1

PLANES DE PERIODOS Y PRODUCTOS

Hay dos clases de planificación. La primera está basada en un período de tiempo. Puede ser cualquier intervalo del calendario: un día, semana, mes o año. Un plan del período hace referencia a la forma de planificar la utilización del tiempo durante ese período. La segunda clase de plan está basada en la actividad, cómo desarrollar un programa o escribir un informe. Los productos pueden ser tangibles como los programas o informes, o intangibles como el conocimiento que adquieres al leer un libro o el servicio que proporcionas cuando trabajas en una oficina.

Para ver la diferencia entre planificación del período y del producto, considera la actividad de leer este libro. Para planificar este trabajo, en primer lugar deberías estimar el tiempo total del trabajo. Por ejemplo, puedes suponer que necesitas 20 horas para leer los 20 capítulos del libro. Para el plan del producto, deberías planificar el tiempo que dedicas a la

lectura, por ejemplo una hora a la semana. El plan del producto para esta tarea tendría el objetivo de leer los 20 capítulos del libro en 20 horas. El plan del período sería la forma de repartir el tiempo de lectura en incrementos semanales de una hora.

LA RELACIÓN ENTRE LOS PLANES DEL PERÍODO Y DEL PRODUCTO

Todos nosotros utilizamos el plan del período y el del producto en nuestra vida diaria. En los negocios, por ejemplo, los dos están relacionados tal y como se muestra de forma simplificada en la Figura 4.1. La parte izquierda de la figura trata de las tareas basadas en el producto y la parte derecha trata sobre las tareas basadas en el período. Las dos están relacionadas como veremos a continuación.

La gestión corporativa proporciona fondos para que los departamentos de ingeniería y fabricación desarrollen y produzcan productos. La ingeniería desarrolla productos y los envía a fabricación. Por medio del grupo de marketing, fabricación entrega estos productos a los clientes, los cuales pagan un precio. Ingeniería y fabricación también proporcionan planes de productos a finanzas y administración, que utilizan los planes del producto para producir planes de período para ingresos y gastos anuales y trimestrales. Finanzas y administración envían sus planes a la gestión corporativa. También dan información sobre precios y previsiones a ingeniería y fabricación para que sepan cuántos productos pueden hacer cada mes y qué cobrar a los clientes. La gestión corporativa decide qué dividendos o intereses van a pagar a los inversores y qué nuevas inversiones se necesitan.

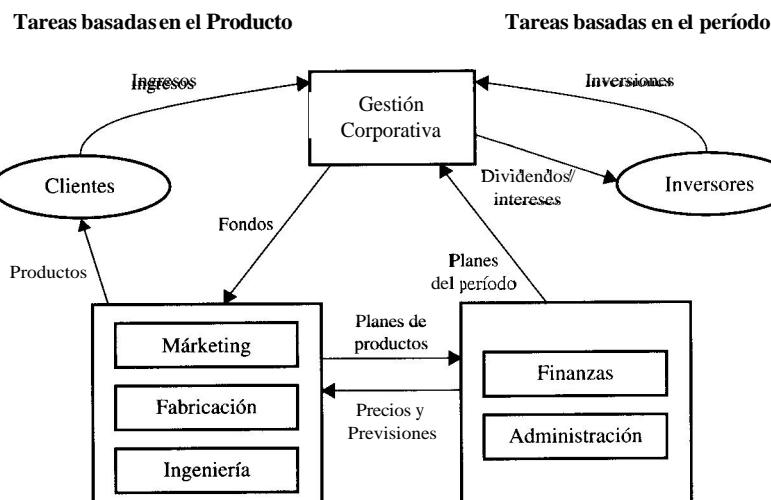


Figura 4.1 Plan del período y del producto.

Cuando sepan cuánto dinero van a tener en el futuro, la gestión corporativa puede decidir que fondos va a proporcionar a ingeniería para el desarrollo y fabricación de futuros productos, completando de esta forma el ciclo.

LA IMPORTANCIA DE LOS PLANES DEL PERÍODO Y DEL PRODUCTO

En los negocios, los planes del producto son importantes porque le indican a marketing cuándo pueden esperar nuevos productos para venderlos a los clientes. También le indican a finanzas lo que costará el desarrollo y la fabricación. Finanzas puede entonces determinar los precios que ellos necesitan cargar y proporcionan unos planes de período precisos para la gestión corporativa. La gerencia puede asignar el dinero que ingeniería y fabricación necesitan. Si los planes del producto no son exactos, los planes del período serán también inexactos. Gerencia, marketing, los inversores y los clientes estarán mal informados. Esto significaría que los fondos necesarios no estarían disponibles para cuando ingeniería y fabricación los necesitasen. Sin los fondos adecuados, ingeniería y fabricación, tendrían que reducir sus planes. Si redujesen sus planes, a ti te podrían despedir.

Aunque este ciclo de planificación puede sonar teórico y necesite varios años, la amenaza es real. La clave para la seguridad del trabajo en cualquier negocio es la salud financiera de la organización. Y una clave para la salud financiera de los negocios es la exactitud de los planes del período y del producto. Por lo tanto es importante que los ingenieros sepan cómo hacer buenos planes del período y del producto.

Aunque los planes del período y del producto deben estar relacionados, son diferentes. Tu trabajo de planificación, tus ingresos y tus otras actividades diarias están gobernadas por períodos de tiempo. Es decir, tú duermes, comes, trabajas y descansas durante ciertos períodos del día o de la semana. Tus gastos e ingresos están gobernados por planificaciones semanales, mensuales y anuales. De esta manera vives en un mundo con períodos de actividades, reglas y limitaciones. El principal propósito de tu trabajo es producir productos y servicios de valor para otros. El coste, la planificación y la calidad de esos bienes y servicios son lo más importante. No puedes hacer un plan competente de uno de ellos sin planificar también el otro. Este capítulo se centra sobre la planificación del período, en el siguiente capítulo discutiremos la planificación del producto.

4.2

EL RESUMEN SEMANAL DE ACTIVIDADES

Para hacer un plan del período, es importante entender cómo gastas tu tiempo. El primer paso es registrar tu tiempo utilizando el Cuaderno de

Registro de Tiempos introducido en el Capítulo 3. Después de reunir los datos de tiempo durante una o dos semanas, empezarás a ver cómo empleas el tiempo. Puesto que los registros de tiempos son muy detallados para el propósito de la planificación, necesitas resumir los datos de una forma más útil. El Resumen Semanal de Actividades de la Tabla 4.1 muestra **los** formatos de tiempo que son más adecuados para la planificación del período.

En las líneas 1 a la 10 del Resumen Semanal de Actividades, registras el tiempo que dedicas a cada actividad principal durante cada día de la Última semana. En la parte inferior derecha, en las líneas 13 a la 16, están la media, el máximo y el mínimo tiempo que dedicas a cada tarea durante las semanas anteriores del semestre. De las líneas 18 a la 21 se

Tabla 4.1 Resumen semanal de actividades.

11 Tiempos y Medias del Período Número de Semanas (número anterior +1): _____

12 Resumen de las semanas anteriores

17 Resumen incluyendo la última semana

18	Total							
19	Avg.							
20	Max.							
21	Min.							

muestran el total, la media, máximo y mínimo del tiempo que dedicas a cada trabajo definido para todo el semestre, incluyendo la Última semana.

Cuando deseas hacer un plan del período para la próxima semana, comienza con el Último Resumen Semanal de Actividades. Basándonos en los tiempos previos dedicados a cada tarea, puedes juzgar cuánto tiempo dedicarás a estas tareas la próxima semana. La forma más sencilla de hacer dicho plan, sería asumir que dedicarás en el futuro la misma cantidad de tiempo que el tiempo medio dedicado en el pasado. Una aproximación más sofisticada debería considerar el trabajo a realizar en la próxima semana y juzgar dónde ubicarte entre los tiempos máximos y mínimos de las semanas anteriores. Esto dará como resultado un plan más exacto. Las siguientes secciones describen cómo llenar el Resumen Semanal de Actividades.

4.3

EL RESUMEN DE LOS TIEMPOS SEMANALES

Un ejemplo del Resumen Semanal de Actividades parcialmente lleno se muestra en la Tabla 4.2. Estos cálculos utilizan datos de la Tabla de Registro de Tiempos mostrados en la Tabla 4.3. Rellena la tabla de la siguiente manera:

1. Anota tu nombre y fecha.
2. Anota las categorías de actividades como cabeceras de las columnas. En el ejemplo de la Tabla 4.2, las categorías de trabajo son: Clases, Codificar Programas, Preparar Exámenes y Leer Textos.
3. Escribe la fecha en la columna Fecha, anotando el domingo como primer día de la semana (fila 3).
4. Para cada día, suma todos los minutos para cada actividad de los registros de tiempos y anota el resultado en la columna correspondiente. Por ejemplo, en la Tabla 4.2 la primera tarea es Clase. En el Cuaderno de Registro de Tiempos de la Tabla 4.3, la Estudiante Y dedicó 50 minutos, desde las 9:00 a las 9:50 A.M. del 9/9, a asistir a clase. Así, puso en la Tabla 4.2 un valor de 50 en la fila 4 en la columna Clase para el día 9/9. El valor total de 96 minutos para Codificar Programas se obtiene a partir de los valores 38 y 58 minutos. Por ello, puso en la Tabla 4.2 el valor 96 en la fila 4 columna Codificar Programas.
5. Después de completar todas las casillas para cada día, se obtiene el total para cada día en la columna Total. Por ejemplo, los totales de la fila 4 de la Tabla 4.2 son $50 + 96 + 80 = 226$.
6. Repite este proceso para cada día de la semana (filas 5-9).

Tabla 4.2 Ejemplo de resumen semanal de actividades.

	Nombre	Estudiante Y						Fecha	9/16/96
1	Tarea	Clases	Codificar	Prep.	Leer				Total
2	Fecha		Progr.	Exam.	Textos				
3	D 8/9								
4	L	50	96		80				226
5	M		62						62
6	Mi	50	69		28				147
7	J		114						114
8	V	50			38				88
9	S			134					134
10	Totales	150	341	134	146				771

11 Tiempos y Medias del Período Número de Semanas (número anterior +1): 1

12 Resumen de las semanas anteriores

13	Total								
14	Med.								
15	Máx.								
16	Mín.								

17

18	Total	150	341	134	146				771
19	Med.	150	341	134	146				771
20	Máx.	150	341	134	146				771
21	Mín.	150	341	134	146				771

7. Calcula los totales semanales para cada tarea clasificada en la fila 10. Por ejemplo, bajo la columna Codificar Programas pondrías $96 + 62 + 69 + 114 = 341$.
8. A continuación, se obtiene el total de los tiempos de las actividades que aparecen en la fila 10. En este caso, el total es $150 + 341 + 134 + 146 = 771$.
9. Por Último, suma los totales diarios para comprobar que obtienes el mismo resultado. En este caso tenemos: $226 + 62 + 147 + 114 + 88 + 134 = 771$, así no hay error. Si dichos números difieren, deberías de volver a calcular el total de todos los días y de todas las tareas para localizar el error.

- 10.** El Resumen de Tiempos de la semana actual se completa tal y como se explicó en la Sección 4.2.

Una buena forma de minimizar los errores es hacer la suma de las columnas dos veces, de arriba abajo y de abajo a arriba, para asegurarte que obtienes el mismo total. Otra forma de minimizar los errores de cálculo podría ser utilizar la hoja de cálculo EXCEL, proporcionada como parte del material de apoyo disponible para este libro¹.

4.4

CÁLCULO DE LOS TIEMPOS Y MEDIAS DEL PERÍODO

Un resumen sencillo de tus tiempos semanales sería suficiente si estuvieras únicamente interesado en una semana, pero realmente estás interesado en los tiempos medios, máximos y mínimos dedicados a estas tareas durante un semestre o año completo. Para mostrar cómo obtener estos datos, completamos a continuación el Resumen Semanal de Actividades mostrado en la Tabla 4.4. Utilizamos el Resumen Semanal de Actividades de la Estudiante Y para la semana 1, mostrado en la Tabla 4.2. Utilizando estos datos, la sección de Tiempos y Medias del Período de la Tabla 4.4 se completa como vemos a continuación:

- 1.** Anota el total de semanas transcurridas. Comprueba la fila 11 del resumen semanal de actividades de semanas anteriores para ver cuántas semanas se tuvieron en cuenta. Puesto que la Tabla 4.2 tiene un 1 en la línea 11, escribe $1 + 1 = 2$ en el número de semanas de la Tabla 4.4.
- 2.** En las filas 13-16 de la tabla de la semana actual (Tabla 4.4), copia todas las entradas de las filas 18-21 de la tabla de la semana anterior (Tabla 4.2).
- 3.** Suma los tiempos dedicados a cada tarea en la semana actual. En cada columna de la fila 18 de la Tabla 4.4 anota el resultado de sumar las filas 13 y 10. Para la primera columna, Clase, dicho resultado es $150 + 150 = 300$. Para la segunda columna, Codificar Programas, obtenemos $339 + 341 = 680$. Haz lo mismo para el resto de columnas. Cada columna de la fila 18 muestra ahora el tiempo total dedicado a cada tarea en las dos primeras semanas de este semestre.
- 4.** Calcula el tiempo medio dedicado semanalmente a cada tarea durante este semestre. En la fila 19 de la Tabla 4.4, divide el valor de cada columna de la fila 18 por el número de semanas. El primer

¹ Las descripciones del material de apoyo y las instrucciones para conseguirlo aparecen en la Última página del libro.

Tabla 4.3 Ejemplo de cuaderno de registro de tiempos para la primera semana.

Estudiante Estudiante Y Fecha 9/9/96
Profesor Sr. Z Clase CS1

Tabla 4.4 Tiempos y medias del período, segunda semana.

	Nombre	Estudiante Y		Fecha	9/23/96
1	Tarea	Clases	Codificar	Prep.	Leer
2	Fecha		Progr.	Exam.	Textos
3	D	15/9			
4	L	50	93		80
5	M		95		
6	Mi	50			71
7	J		77		
8	V	50	74		40
9	S			33	
10	Totales	150	339		224
					713

11 **Tiempos y Medias del Período Número de Semanas (número anterior +1):** 2**12 Resumen de las semanas anteriores**

13	Total	150	341	134	146					771
14	Med.	150	341	134	146					771
15	Máx.	150	341	134	146					771
16	Mín.	150	341	134	146					771

17 Resumen incluyendo la Última semana

18	Total	300	680	134	370					1484
19	Med.	150	340	67	185					742
20	Máx.	150	341	134	224					771
21	Mín.	150	339	134	146					713

valor medio es de $300/2=150$ y el segundo es $680/2=340$. Repite estos cálculos para el resto de columnas.

5. Para encontrar el mayor tiempo dedicado a una tarea durante una semana, calcula el tiempo máximo. Calcula el valor Máximo en la fila 20, comparando cada columna de la fila 15 con la misma columna de la fila 10. Anota el valor mayor en la fila 20. Para la columna leer Textos, el valor Máximo anterior era 146 (véase fila 15) y el valor para la semana actual es de 224 (véase fila 10). Puesto que el valor mayor es 224, anótalo en la fila 20.
6. Para saber el menor tiempo que has dedicado a una tarea durante una semana, calcula el tiempo mínimo. Para calcular el valor

Mínimo de la columna 21, compara cada columna de la fila 16 con la misma columna de la fila 10. Anota el valor más pequeño en la fila 21. Para la columna Leer Texto, el valor anterior Mínimo era 146 (véase fila 16) y el valor para la semana actual es 224 (véase fila 10). Puesto que 146 es menor, anótalo en la fila 21. La fila 21 muestra los tiempos mínimos dedicados cada semana a cada actividad. Observa que ignoras los valores nulos en el cálculo de los tiempos mínimos.

7. Observa que, excepto para la primera semana, el Tiempo Total Máximo y Mínimo serán diferentes.

La columna Preparar Exámenes proporciona un ejemplo de por qué deberías ignorar los valores nulos en el cálculo de los tiempos mínimos. Puesto que no se dedicó tiempo a esta actividad en esta semana, el tiempo medio tiene un valor de 67 minutos. Así, la Estudiante Y dedicó 134 minutos una semana y cero la siguiente, dando un valor medio 67 minutos. El valor Máximo se calcula como hemos visto antes, pero el valor Mínimo cero no es Útil. La razón es que para la actividad de Preparar Exámenes ya sabes que no le dedicarás tiempo durante muchas semanas del semestre. Así el valor cero para el Mínimo no añade nueva información. Puede ser interesante, sin embargo, saber que cuando estudiaste para los exámenes, el tiempo mínimo que dedicaste fue de 134 minutos. Puesto que solamente quieres saber la cantidad mínima de tiempo dedicado cuando hiciste esta tarea, sustituye el valor Mínimo en la fila **16** por un valor mayor de cero en la fila 10.

Las orientaciones para completar el Resumen Semanal de Actividades aparecen en las Tablas 4.5 y 4.6.

4.5

LA UTILIZACIÓN DEL RESUMEN SEMANAL DE ACTIVIDADES

Para completar esta Tabla, cada semana tendrás que calcular el tiempo medio, máximo y mínimo que has dedicado a cada actividad cada semana. Aunque esta tabla puede parecer un poco complicada, después de un poco de práctica, será inmediata. Verás que es una forma sencilla de organizar y conservar una gran cantidad de datos de tiempos. Si utilizas una hoja de cálculo para hacer estos cálculos, será aún más fácil.

Los datos en el Resumen Semanal de Actividades te ayudarán a entender cómo gastas tu tiempo. Con esta información, por ejemplo, podrías estimar que una gran tarea probablemente se realizará en el tiempo máximo y una tarea sencilla se realizará en el tiempo mínimo. Puedes utilizar estos datos para planificar las semanas subsiguientes. La planificación se discute en los capítulos siguientes.

Tabla 4.5 Instrucciones resumen semanal de actividades
 (1) Resumen semanal del tiempo.

Propósito	Esta tabla se utiliza para controlar y analizar el tiempo gastado durante este curso. Estos datos se obtienen del Cuaderno de Registro de Tiempos.
Método	Resume los datos del Cuaderno de Registro de Tiempos al final de cada semana. Si la clasificación de las tareas no es adecuada, cámbialas. Si no tienes suficientes columnas para todas las tareas, utiliza copias de la tabla.
Cabecera	Introduce los siguientes datos: Tu nombre. La fecha actual.
Tarea	Anota los nombres de las principales tareas a las que dedicarás tu tiempo durante este curso.
Ejemplo	Clases, Codificar Programas, Revisar exámenes, Leer libros, etc.
Fecha	Con D (Domingo), anota la fecha para ese día.
Ejemplo	819
Columnas	Para cada día de la semana, determina el tiempo total dedicado a cada tarea descrita en el Cuaderno de Registro de Tiempos. Anota este número en la columna correspondiente para ese día.
Ejemplo	Para el lunes, 919, los tiempos de Codificar Programas del Cuaderno de Registro de Tiempos eran 38 y 58 minutos. Anota la suma (96 minutos) en la fila 4 del lunes 919.
Totales Semana (fila 10)	Anota en la fila 10 el total para cada tarea durante una semana completa.
Ejemplo	$96+62+69+114=341$ en la fila 10 columna Codificar Programas.
Totales	Para cada fila, anota el total de los tiempos de las tareas para un día dado, columna de la derecha. Si tienes varias tablas, anota el total de todas las tablas en la columna situada más a la derecha (en la primera tabla).
Ejemplo	$50+96+80=226$ para el Total del Lunes 919
Primera Comprobación	Calcula el total de los tiempos dedicados durante una semana y anótalos en la columna y fila de Total para esa semana.
Ejemplo	$226+62+147+114+88+134=771$ es el total semanal.
Comprobación Final	$150+341+134+146=771$ total de las tareas (fila 10). Puesto que es igual al total de la columna de la derecha no hay error.

Aunque al principio aprenderás mucho del Resumen Semanal de Actividades, pronto suministrará poca información cada semana. Cuando llegues a este punto, prepara un Resumen Semanal de Actividades una vez

Tabla 4.6 Instrucciones resumen semanal de actividades
 (2) Resumen de medias, máximos y mínimos.

Propósito	Esta tabla se utiliza para controlar y analizar el tiempo dedicado a este curso. Estos datos se obtienen del Cuaderno de Registro de Tiempos.
Método	Estas instrucciones son una continuación de las vistas en las Instrucciones Resumen Semanal de Actividades (1). Pasa la cabecera y el resto de información consulta las instrucciones de la Tabla 4.5.
Número de semanas (fila 11)	Este es el número de semanas incluidas en el resumen de datos. Consulta el Número de Semanas en la fila 11 para ver cuantas semanas previas se han acumulado. Suma uno a dicho número, y anota el total.
Tiempos Hasta la Fecha (filas 13-16)	Para las filas 13 a 16, copia, columna a columna, los datos de las filas 18 a 21 de los resúmenes de semanas anteriores. Incluye las filas de Total, Media, Máx. y Mín. para cada columna, incluyendo la columna total.
Total Tiempos Actuales (fila 18)	Para calcular los valores de la fila 18, suma las cantidades de las filas 10 y 13. Haz esto para cada columna de esta tabla.
Ejemplo	En la Tabla 4.4, el valor de la columna Codificar Programas para la fila 18 se calcula así, $339+341=680$.
Media Tiempos Actuales (fila 19)	El valor medio se calcula dividiendo los valores de la fila 18 en cada columna por el número de semanas de la fila 11.
Ejemplo	En la Tabla 4.4, el valor medio de Codificar Programas es $680/2=340$.
Máximo Tiempo Actual (fila 20)	Los valores de la fila 20 se obtienen comparando los valores de la fila 15 con los de la fila 10. Anota el valor mayor de los dos números. Haz esto para cada columna, incluyendo la columna Total.
Ejemplo	En la Tabla 4.4 y para Codificar Programas, el valor Máximo era 341 y el valor para esta semana es 339, por lo que el valor Máximo sigue siendo 341.
Otro Ejemplo	En la Tabla 4.4 para Leer Libros, el valor Máximo era 146 y el valor para esta semana es 224, por lo que el nuevo valor máximo es 224.
Mínimo Tiempo Actual (fila 21)	Los valores de la fila 21 se obtienen comparando los valores de las filas 16 y 10. Anota el valor más pequeño superior a cero. Haz esto para cada columna, incluyendo la columna Total.
Ejemplo	En la Tabla 4.4 y para la actividad Codificar Programas, el valor Mínimo era 341 y el valor para la fila 10 es 339, el nuevo valor Mínimo sigue siendo 339.
Otro Ejemplo	En la Tabla 4.4 y para la actividad Leer Libros, el valor Mínimo era 146 y el valor para la fila 10 es 224, el nuevo valor Mínimo sigue siendo 146.
Un ejemplo más	En la Tabla 4.4 y para la actividad Preparar Exámenes, el valor Mínimo era 134 y el valor para la fila 10 es 0, el nuevo valor Mínimo sigue siendo 134.

al mes o comprueba que la distribución actual de tiempos es la adecuada. También, una vez que hayas reunido los datos de proyectos en el Capítulo 5 y posteriores, sabrás cuánto tiempo dedicas a los proyectos y podrás reunirlos en el Resumen Semanal de Actividades cuando lo necesites.

En la práctica he pasado a utilizar un Resumen Mensual de Actividades para controlar el tiempo que he dedicado a mis dos principales actividades: escribir libros y realizar otros proyectos. Puesto que escribo libros en mi tiempo particular y hago proyectos en mi tiempo en la empresa, es importante saber cuánto tiempo particular y de proyectos puedo esperar tener disponible durante cualquier semana o mes. Esto me ayuda a hacer los planes con precisión y cumplir compromisos. Para este propósito, una revisión mensual es suficiente. Sigo encontrando útil resumir mi tiempo cada mes.

RESUMEN

Este capítulo muestra como la planificación de períodos y de productos están relacionados y describe la planificación del período. El primer paso en la planificación del período es entender tus datos de tiempos. Comienza con datos de tiempos básicos a partir del Cuaderno de Registro de Tiempos y haz un Resumen Semanal de Actividades para cada semana. El Resumen Semanal de Actividades proporciona un registro del tiempo dedicado a cada actividad y el tiempo medio, máximo y mínimo dedicado a cada actividad durante un semestre. Finalmente, puedes dejar de resumir los tiempos semanales y hacer un resumen cuando tu distribución de tiempos cambie o consideres interesante resumir mensualmente tus tiempos.

EJERCICIO 4

Completa y entrega el Resumen Semanal de Actividades para las dos semanas anteriores. Utiliza los registros diarios de tiempos para llenar esta tabla. Entrega una copia de las páginas de tiempos registrados que no hayas entregado todavía.

CAPITULO 5

La planificación de/ producto

Este capítulo describe cómo utilizar los datos de los registros de tiempos para hacer planes del producto. La nueva tabla, el Cuaderno de Trabajos se introduce para ayudar a registrar los datos históricos del producto. También se muestra un ejemplo de cómo llenar la tabla. En cuanto al ejercicio, consistirá en llenar el Cuaderno de Trabajos con las tareas que hayas hecho hasta ahora en el curso.

5.1 LA NECESIDAD DE LOS PLANES DEL PRODUCTO

Hace algunos años, cuando trabajaba en IBM, estaba encargado del departamento de desarrollo de grandes proyectos software. Muchos de los proyectos se retrasaban bastante y el director estaba preocupado. Mi trabajo consistió en arreglar la complicada situación. La primera cosa que hice, fue revisar los proyectos principales. Me sorprendí de que ninguno tuviese planes documentados. Inmediatamente, insistí en que todos los ingenieros hiciesen planes para todos sus proyectos. Ninguno de ellos había preparado anteriormente un plan formal, y les llevó un par de meses hacerlo. Se impartieron clases sobre planificación de proyectos, ellos hicieron sus planes, y entonces pudimos establecer programaciones realistas para su trabajo.

El acto de hacer planes tiene un efecto sorprendente: este grupo de desarrollo nunca había entregado anteriormente un producto a tiempo. Con sus nuevos planes, no se equivocaron en una fecha durante los dos años y medio siguientes. Desde entonces, han estado planificando su trabajo. La planificación es una parte crítica del trabajo del ingeniero del software, y por lo tanto para ser un ingeniero del software, tienes que saber cómo hacerla. La clave está en la práctica, para adquirirla, comienza haciendo planes ya y continúa haciéndolos en todos tus proyectos futuros.

5.2

POR QUÉ SON ÚTILES LOS PLANES DEL PRODUCTO

En este libro, sugiero que desarrolles planes de producto para todos tus proyectos o tareas más importantes: codificar un programa, leer un libro o preparar un informe. El plan del producto te ayudará a decidir cuánto tiempo necesitas para hacer tu trabajo y cuándo lo acabarás. Los planes, también ayudan a controlar el progreso mientras se está haciendo el trabajo.

Cuando los ingenieros trabajan en equipos de desarrollo, necesitan planificar su trabajo personal. La planificación proporciona una sólida base para comprometerse a unas fechas de entrega, y permite a los ingenieros coordinar sus trabajos con los productos. Sus planes individuales del producto, les permiten cumplir las fechas para cada una de sus tareas interdependientes y cumplir sus compromisos de forma consistente.

En los negocios se utilizan los planes del producto por las mismas razones que lo vas a hacer tú: para planificar y gestionar el trabajo. Un plan bien hecho incluye una estimación del coste del proyecto. Las estimaciones son esenciales para contratar desarrollos, ya que los clientes necesitan saber el precio por adelantado. Las estimaciones también son necesarias cuando se desarrollan productos. El coste del proyecto es la parte más importante del precio de un producto y debe ser lo bastante bajo para que el precio sea competitivo en el mercado.

Los ingenieros también utilizan los planes del producto para entender el estado de su proyecto. Con planes razonablemente detallados y precisos, pueden juzgar dónde se encuentra el proyecto frente al plan. Puede ser, que estén retrasados y necesiten ayuda, o que tengan que retrasar la programación. Pero también pueden ir más adelantados con respecto a lo planificado, y ser capaces de ayudar a sus compañeros o entregar antes. Cuando los ingenieros hacen planes pueden organizar mejor su tiempo, y evitar las crisis en los últimos minutos. Entonces es menos probable que cometan errores y generalmente producirán mejores resultados.

Puesto que la planificación es tan importante, necesitas conocer cómo hacer planes precisos. Y también, saber compararlos con tus resultados reales, para que puedas aprender a hacer mejores planes.

5.3**¿QUÉ ES UN PLAN DEL PRODUCTO?**

El primer paso para hacer un plan del producto, es tener una definición clara del producto que quieras producir. Aunque esto parece obvio, es sorprendente como con frecuencia la gente se sumerge en los mecanismos de desarrollo del producto antes de definir qué están intentando hacer. Únicamente después de saber qué quieres hacer, podrías comenzar a pensar cómo hacerlo. En este momento comienza la planificación.

Un adecuado plan del producto requiere tres cosas:

- El tamaño y las características más importantes del producto a realizar.
- Una estimación del tiempo requerido para hacer el trabajo.
- Una previsión de la planificación.

El producto podría ser un programa ejecutable, un diseño de un programa o un plan de pruebas. Un plan del producto identifica el producto a hacer y contiene estimaciones del tamaño del producto, las horas de trabajo a realizar y la programación. Productos más complejos requieren una planificación más sofisticada y más tipos de información, tales como asignación de responsabilidades, planes de personal, especificaciones de procesos o productos, dependencias con otros grupos, pruebas especiales y cuestiones de calidad. En este libro, sin embargo, trataremos solamente los tres elementos básicos del plan: estimaciones de tamaño, horas previstas y programaciones.

5.4**LA PLANIFICACIÓN DEL PRODUCTO EN ESTE LIBRO .**

La planificación del producto es una habilidad que puede mejorarse con la práctica. Si como estudiante empiezas ahora a hacer planes, desarrollarás tus habilidades de planificación. Así, cuando necesites elaborar un plan para un proyecto importante, sabrás cómo hacerlo. En este capítulo, deberías hacer un plan para cada tarea principal. Y en el Capítulo 10, sabrás hacer planes más globales para desarrollar programas.

En este libro, empezamos planificando pequeños trabajos, porque esta es una buena manera de aprender a planificar. Después de todo, si no puedes planificar una tarea pequeña, ¿cómo podrás planificar un gran proyecto? Los ingenieros en su trabajo hacen muchas tareas pequeñas como parte de un gran trabajo. Así, si planificas cada una de estas pequeñas tareas, tendrás más oportunidades de desarrollar tus habilidades de planificación. En grandes trabajos, puedes integrar estos pequeños planes en un gran plan, para el trabajo completo. Esta es la forma más efectiva de hacer planes exactos para grandes trabajos.

5.5**LA PLANIFICACIÓN DE PEQUEÑOS TRABAJOS**

Es importante hacer un plan que se aadecue a la magnitud y complejidad del trabajo a realizar. Tendría poco sentido, por ejemplo, hacer un plan sofisticado para una tarea que necesitara una o dos horas. A la inversa, con una tarea que necesitase varias semanas, un plan más elaborado está justificado.

El plan del producto más básico consistirá únicamente en estimar el tiempo necesario para hacer una tarea o trabajo. Una vez que puedes estimar con precisión el tiempo para hacer el trabajo, todas las otras cuestiones de planificación se pueden hacer de una forma bastante fácil.

Reuniendo datos sobre la duración de varias tareas realizadas anteriormente, puedes predecir para unas tareas de duración similar, cuánto tiempo necesitarán en el futuro. Esta aproximación se puede aplicar a casi cualquier tarea relacionada con un producto que necesite de unas pocas horas a unos pocos días para su realización. Por ejemplo, si planificas leer un capítulo de un libro, sería útil saber cuánto tiempo necesitarás en leer los capítulos anteriores. Con los datos de los tiempos de lectura medio, máximo y mínimo, puedes predecir mejor tu tiempo de lectura para un nuevo capítulo.

5.6**ALGUNAS DEFINICIONES**

Antes de continuar, necesitamos definir algunos términos:

- Un **producto** es algo que produces para un compañero, un empresario o un cliente.
- Un **proyecto** normalmente produce un producto.
- Una **tarea** se define como un elemento de trabajo.
- Un **proceso** se define como la forma de hacer proyectos. El Capítulo 11 trata de una forma más detallada **los procesos**.
- Los **planes** describen la forma en que un proyecto concreto va a ser hecho: cómo, cuándo y qué coste tendrá. También puedes planificar tareas individuales.
- Un **trabajo** es algo que haces, tanto un proyecto como una tarea.

5.7**EL CUADERNO DE TRABAJOS**

El Cuaderno de Trabajos que **se** muestra en la Tabla 5.1 se diseñó para registrar los datos de tiempos estimados y reales. Obsérvese que este cuaderno es un documento de planificación del producto, ya que trata datos

del producto. A la inversa, el Cuaderno de Registro de Tiempos y el Resumen Semanal de Actividades contienen datos de planificación de períodos.

Las entradas en el Cuaderno de Trabajos se hacen como se muestra en la Tabla 5.2. Este ejemplo utiliza datos del Cuaderno de Registro de Tiempos de la Estudiante Y de la Tabla 5.3. Las instrucciones del Cuaderno de Trabajos se muestran en la Tabla 5.4. Obsérvese que en el Cuaderno, la Estudiante Y controla dos actividades generales: asistir a clases y preparar exámenes. También controla dos actividades importantes: codificar programas y leer capítulos del libro. Las casillas en la Tabla 5.2 se completan de la siguiente manera:

- | | |
|------------------------------|---|
| 1. Trabajo | Cuando estés planificando una actividad, asignale un número de trabajo comenzando por 1. |
| 2. Fecha | Escribe la fecha de comienzo del trabajo. |
| 3. Proceso | Escribe el tipo de tarea, tal como leer un libro, codificar un programa o preparar un informe. |
| 4. Tiempo estimado | Calcula el tiempo que crees que gastarás en hacer esa tarea y anótalo en la columna de Tiempo Estimado. Para hacer esta estimación, examina los datos de proyectos similares anteriores y úivilzalos. Obsérvese que para la estimación del programa 6 (trabajo 10), la Estudiante Y utilizó el tiempo medio de los programas 1-5. Encontró un valor de 105,8 minutos observando las Velocidades Hasta la Fecha de programas anteriores (trabajo 8). |
| 5. Unidades estimadas | Para las unidades elementales de trabajo, escribe un 1 en la columna de Unidades Estimadas. En el próximo capítulo se introducen medidas de tamaño más refinadas. |
| 6. Tiempo real | Al terminar el trabajo, anota el tiempo que le has dedicado. Para el trabajo 10, este tiempo fue de 151 minutos. |

7. Unidades reales

Cuando termines el trabajo, registra las unidades reales. Aquí, debería ser una por cada trabajo. En el capítulo siguiente, se utiliza una medida del tamaño para las unidades de trabajo.

8. Velocidad real

La velocidad real es el Tiempo Real dividido por las Unidades Reales. Para el programa 6 (trabajo 10) la Velocidad Real es $151/1 = 1.51$ minutos por programa.

9. Tiempo Hasta la Fecha

Al acabar el trabajo, calcula y anota el Tiempo Hasta la Fecha para todas las tareas realizadas hasta la fecha para el mismo proceso. Por ejemplo, con 151 minutos para el Tiempo Real para el programa 6 y un Tiempo Hasta la Fecha para los programas 1-5 de 529 minutos (véase trabajo 8), el Tiempo Hasta la Fecha para el programa 6 es $151 + 529 = 680$ minutos.

10. Unidades Hasta la Fecha

Anota las Unidades Hasta la Fecha para todas las tareas de cada tipo que hayas finalizado. Con el programa 6, añade las Unidades Reales para este trabajo a las Unidades Hasta la Fecha para los trabajos más recientes de este tipo (de nuevo, el trabajo 8): $5 + 1 = 6$.

11. Velocidad Hasta la Fecha

La Velocidad Hasta la Fecha es el Tiempo Hasta la Fecha dividido por las Unidades Hasta la Fecha, es decir $680/6 = 113,3$ minutos por programa. Este es el tiempo medio dedicado a hacer un trabajo de este tipo.

12. MÁXIMO HASTA LA FECHA

Para encontrar la velocidad máxima para cualquier tarea, compara la Velocidad Real de los trabajos más recientes con el MÁXIMO Hasta la Fecha de los trabajos previos del mismo tipo y anota el valor. Para el programa 6, el MÁXIMO Hasta la Fecha para el programa 5 (trabajo 8) es 158 minutos por unidad, que es mayor que la Velocidad Real de 151 minutos

por unidad para el programa 6, por lo tanto escribe 158 para el programa 6 en el Mínimo Hasta la Fecha.

13. Mínimo Hasta la Fecha

El valor Mínimo Hasta la Fecha es la velocidad mínima para cualquier tarea de un tipo dado. Para determinar este valor, compara la Velocidad Real del trabajo con el Mínimo Hasta la Fecha de los trabajos más recientes de este tipo y escribe el valor menor. Para el programa 6, por ejemplo, la Velocidad Real de 151 minutos por unidad es mayor que los 69 minutos por unidad para el Mínimo Hasta la Fecha del programa 5 (trabajo 8), por lo que el programa 6 mantiene el valor de 69 en el Mínimo Hasta la Fecha.

Con estos datos puedes consultar la velocidad media para cualquier tarea dada, así como las velocidades máxima y mínima. La ventaja principal del Cuaderno de Trabajos es que proporciona una forma concisa de registrar y acceder a una gran cantidad de datos históricos de proyectos. Como verás, es la clave para hacer estimaciones exactas. Las estimaciones exactas, son a su vez la clave para hacer buenas planificaciones.

5.8

ALGUNAS SUGERENCIAS SOBRE CÓMO UTILIZAR EL CUADERNO DE TRABAJOS

Las siguientes prácticas deberían ayudarte a hacer una utilización más efectiva del Cuaderno de Trabajos:

Para el primer trabajo de un tipo dado, la Estudiante Y no tenía datos previos para guiar sus estimaciones. Ella las tuvo que predecir. Suponer la primera vez está bien, pero conforme vayas reuniendo datos ya no tienes que seguir adivinando.

En general para la estimación del tiempo de un nuevo trabajo, querrás utilizar la Velocidad Hasta la Fecha del trabajo del mismo tipo realizado recientemente. La Estudiante Y hizo esto. Para el trabajo 6, la Estudiante Y no utilizó el valor de 36 minutos por capítulo para la Velocidad hasta la Fecha. Presumiblemente, pensó que el Capítulo 4 le llevaría más tiempo leerlo que la media de los capítulos anteriores. Estaba en lo cierto, y le llevó más tiempo que el esperado. Es importante recordar que los datos del Cuaderno de Trabajos están para ayudarte a hacer planes.

Tabla 5.1 El cuaderno de trabajos.

Nombre: _____ Fecha: _____

Trabajo	Fecha	Proceso	Estimado		Real			Hasta la Fecha					
			Tiempo	Unidades	Tiempo	Unidades	Velocidad	Tiempo	Unidades	Velocidad	Máx.		
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													
Descripción:													

Trabajo	Fecha	Proceso	Estimado			Real			Hasta la Fecha				
			Tiempo	Unidades	Tiempo	Unidades	Velocidad	Tiempo	Unidades	Velocidad	Máx.	Mín.	
1	9/9	Cod.	100	1	158	1	158	158	1	158	158	158	158
<i>Descripción:</i> Escribir el programa 1 (minutos por programa)													
2	9/9	Texto	50	2	80	2	40	80	2	40	40	40	40
<i>Descripción:</i> Leer los Capítulos 1 y 2 del libro de texto (minutos por capítulo)													
3	11/9	Cod.	158	1	69	1	69	227	2	113,5	158	69	
<i>Descripción:</i> Escribir el programa 2													
4	11/9	Texto	40	1	28	1	28	108	3	36	40	28	
<i>Descripción:</i> Leer el Capítulo 3 del libro de texto													
5	12/9	Cod.	114	1	114	1	114	341	3	113,7	158	69	
<i>Descripción:</i> Escribir el programa 3													
6	13/9	Texto	60	1	118	1	118	226	4	56,5	118	28	
<i>Descripción:</i> Leer el Capítulo 4 del libro de texto													
7	16/9	Cod.	114	1	93	1	93	434	4	108,5	158	69	
<i>Descripción:</i> Escribir el programa 4													
8	17/9	Cod.	109	1	95	1	95	529	5	105,8	158	69	
<i>Descripción:</i> Escribir el programa 5													
<i>Descripción:</i> Leer el Capítulo 5 del libro de texto													
10	19/9	Cod.	106	1	151	1	151	680	6	113,3	150	69	
<i>Descripción:</i> Escribir el programa 6													
11	20/9	Texto	59	1	40	1	40	337	6	56,2	118	28	
<i>Descripción:</i> Leer el Capítulo 6 del libro de texto													
12	21/9	Texto	56	1									
<i>Descripción:</i> Leer el Capítulo 7 del libro de texto													
<i>Descripción:</i>													
<i>Descripción:</i>													
<i>Descripción:</i>													
<i>Descripción:</i>													
<i>Descripción:</i>													
<i>Descripción:</i>													
<i>Descripción:</i>													
<i>Descripción:</i>													

Para encontrar rápidamente todas las entradas en el Cuaderno de Registro de Tiempos para un número de trabajo dado, es útil añadir el númer

Tabla 5.3 Ejemplo del cuaderno de registro de tiempos.

Estudiante _____ **Estudiante Y** _____ **Fecha** 9/9/96
Profesor _____ **Sr. Z** _____ **Clase** IP

mero de trabajo en la columna Actividad del Cuaderno de Registro de Tiempos, tal como se muestra en la Tabla 5.3. En efecto, encontrarás más

Tabla 5.4 Instrucciones para el cuaderno de trabajos.

Propósito	Esta tabla se utiliza para controlar los trabajos de cada proyecto. Registra la información clave de cada proyecto. Un proyecto es cualquier actividad que deseas controlar, como desarrollar un programa o escribir un artículo.
Método	Cuando comiences un proyecto, anota el número de trabajo en este cuaderno. Anota los números comenzando desde 1.
Cabecera	Introducelos siguientes datos: Tu nombre. La fecha de comienzo de este Cuaderno de Trabajos.
Trabajo	Anota el número de trabajo que has seleccionado.
Fecha	Anota la fecha de inicio del trabajo
Proceso	Anota el tipo de tarea. Por ejemplo, para un trabajo técnico anota Artículo , para desarrollar un programa, utiliza Cod. , etc.
Tiempo Estimado	Anota el tiempo total en minutos que se estimó para ese trabajo. Utiliza los valores de Velocidad Hasta la Fecha, Máximo y Mínimo como guía. Si estas velocidades no son razonables, utiliza tu experiencia.
Unidades Estimadas	Anota las unidades estimadas para el trabajo acabado. Cuando hagas una tarea, escribe 1 en las Unidades.
Tiempo Real	Anota el tiempo total real para el trabajo realizado.
Unidades Reales	Anota el número real de unidades totales. De nuevo, escribe 1 unidad para cada trabajo acabado.
Velocidad Real	Anota el Tiempo Real dividido por las Unidades Reales.
Tiempo hasta la fecha	Localiza el trabajo del mismo tipo que tu nuevo trabajo. Suma al valor de Tiempo Hasta la Fecha de este trabajo el Tiempo Real para el nuevo trabajo. Anota este total en la casilla de Tiempo Hasta la Fecha para el nuevo trabajo.
Unidades hasta la fecha	Localiza el trabajo del mismo tipo que tu nuevo trabajo. Suma las Unidades Hasta la Fecha de este trabajo a las Unidades Reales para el nuevo trabajo. Anota este total en la casilla de Unidades Hasta la Fecha para el nuevo trabajo.
Velocidad hasta la fecha	Divide el tiempo Hasta la Fecha por las Unidades Hasta la Fecha para obtener los minutos por unidad para todos los trabajos terminados hasta la fecha. Anota este número en la casilla Velocidad hasta la Fecha, para este trabajo.
Máx.	Anota la velocidad máxima para todos los trabajos de cada tipo que hayan acabado.
Mín.	Anota la velocidad mínima para todos los trabajos de cada tipo que hayan acabado.
Descripción	Anota una descripción del trabajo que se ha hecho. Sé suficientemente claro para que el contenido del trabajo pueda ser fácilmente identificado.

adecuado dejar todos los tipos de actividades y referenciarlas por sus números. Seguiremos esta práctica en los ejemplos restantes de este libro.

Después de aprender a utilizar el Cuaderno de Trabajos, probablemente encontrarás más adecuado utilizar una hoja de cálculo para hacer los cálculos. Un ejemplo de hoja de cálculo de este tipo se incluye en el material de apoyo de este libro. Las instrucciones para conseguir dicho material se indican en la Última página del libro.

5.9

LA UTILIZACIÓN DE TIEMPOS Y MEDIAS DEL PRODUCTO

Supongamos que tienes los datos de la Tabla 5.2 y quieres planificar tu tiempo para la próxima semana. Puedes, por ejemplo, planificar, codificar dos programas y leer cuatro capítulos. Puedes saber que uno de los programas es más complejo que cualquiera de los que hayas codificado hasta ahora y el otro se parece a la media.

Para estimar los tiempos para codificar estos dos programas, observa en la Tabla 5.2 el trabajo 10. Aquí, el tiempo medio para codificar los seis primeros programas es de 113,3 minutos, un poco menos de 2 horas. El tiempo máximo ha sido 158 minutos o aproximadamente 2 horas y media. Para un programa medio es probable que se necesiten 2 horas y para el más complejo alrededor de 158 minutos o más. Para estar más seguro, supón que el programa complejo necesitará 3 horas y media, o sea 210 minutos. Esto da un total de 330 minutos de programación para la próxima semana.

Para los capítulos del libro, consideras que dos capítulos serán de tipo medio y los otros dos necesitarán un poco más de tiempo. De los datos del Trabajo 11, los tiempos medios para leer capítulos han sido de 56,2 minutos. Puedes asumir que leerás los dos capítulos intermedios en una hora cada uno. Para los capítulos más complejos, sin embargo, necesitarás el tiempo máximo de 118 minutos, unas dos horas por capítulo.

En total, prevés dedicar durante la próxima semana unas 5 horas y media a codificar programas y 6 horas para leer el libro. Con esta información, puedes mejorar la planificación para estos trabajos.

Como verás pronto, los tiempos para las tareas pequeñas variarán considerablemente tus estimaciones. Esto es normal. La clave para una buena estimación, sin embargo, es aprender a hacer estimaciones equilibradas. Un conjunto equilibrado de estimaciones tiene en cuenta las sobreestimaciones y las subestimaciones. En el ejemplo de la Tabla 5.2, 5 de las 11 estimaciones fueron bajas, 5 altas y el trabajo 5 se hizo en el tiempo previsto. Las estimaciones bajas fueron las de los trabajos 1, 2, 6, 9 y 10, las otras cinco estimaciones fueron altas. La Estudiante Y había aprendido a hacer estimaciones equilibradas. Hizo esto, utilizando sus ve-

locidades medias anteriores para estimar un nuevo trabajo. Mientras hagas esto con coherencia y tus trabajos sean razonablemente similares, pronto harás estimaciones equilibradas.

La ventaja de hacer estimaciones equilibradas es que, en promedio, tus trabajos durarán aproximadamente lo que hayas estimado. Es decir, cuando un trabajo se realice en más tiempo, se compensará con otro que necesite menos tiempo. Cuando estés estimando grandes trabajos como agrupaciones de muchos trabajos pequeños, es más probable que consigas una aproximación razonable al plan global.

RESUMEN

Los planes del producto te ayudan a resumir y gestionar datos de tiempos para los productos que planificas hacer: codificar un programa, redactar informes, leer capítulos de libros y estudiar para los exámenes. El Cuaderno de Trabajos es una herramienta para controlar el tiempo dedicado a cada tarea. Los cálculos del Cuaderno de Trabajos utilizan datos de tiempos del Cuaderno de Registro de Tiempos.

Cuando lo completes, el Cuaderno de Trabajos te proporciona **los** tiempos medio, máximo y mínimo que dedicas a terminar cada uno de tus proyectos hasta ese momento. Puedes utilizar esta información para planificar el trabajo de la próxima semana o para hacer planes más sofisticados que se requieran en capítulos posteriores.

EJERCICIO 5

Completa y entrega el Cuaderno de Trabajos para el trabajo que hayas hecho hasta ahora en el curso. Donde hayas estimado datos anótalos, de lo contrario, deja el espacio de estimación en blanco. Utiliza los cuadernos diarios de tiempos para completar estas tablas. Continúa hasta completar los datos en el Cuaderno de Trabajos para cada proyecto. Haz una estimación del tiempo y de las unidades antes de iniciar un trabajo, registra los datos reales y los datos Hasta la Fecha cuando hayas finalizado.

CAPITULO 6

El tamaño del producto

Hasta ahora hemos considerado grandes unidades de trabajo, como programas o capítulos de un libro. Para hacer planes del producto, necesitas utilizar medidas más precisas. Este capítulo describe cómo medir y estimar el tamaño del producto. En el ejercicio del capítulo, medirás el tamaño de los trabajos que has hecho en los ejercicios anteriores de este libro y calcularás tu productividad para ese trabajo.

6.1 EL PROCESO DE PLANIFICACIÓN DEL PRODUCTO

La planificación del producto no es un proceso exacto. Aunque hicieses los mejores planes que pudieses, reconoce que estos probablemente serán inexactos. La planificación es una habilidad que se puede desarrollar y mejorar: desarrolla planes para cada trabajo que hagas, y una vez acabado, compara lo planificado con lo que realmente has hecho. Así, entenderás mejor los errores cometidos en estos planes y aprenderás a hacerlos mejor.

Para hacer un plan del producto, compara lo que planificas hacer con lo que has hecho antes. Comienza con datos de trabajos parecidos y compara la nueva tarea con los trabajos anteriores. Busca trabajos previos que hayas hecho del mismo tamaño y planificación. Entonces, basándote en el tamaño de los trabajos que anteriormente hayas realizado, estima el tamaño del nuevo trabajo a realizar.

La razón principal para reunir datos de trabajo en este curso, es para saber qué tiempo dedicas a las distintas tareas que haces. Si, por ejemplo, juzgas que una nueva tarea es parecida a dos tareas realizadas anteriormente, y cuya duración fue de 3 y 4,2 horas respectivamente, probablemente deducirás que la nueva tarea se realizará en un tiempo medio de 3,6 horas. Esta estimación burda, probablemente será mejor que una estimación hecha sin ningún dato histórico.

6.2 MEDIDA DEL TAMAÑO

Puesto que las tareas, a menudo, varían considerablemente en tamaño y complejidad, sería útil tener una forma de comparar sus tamaños. Consideremos la lectura de los capítulos de un libro de texto. Con los datos del tiempo dedicado a leer cinco capítulos, podrías estimar el tiempo de leer el Capítulo 6. Una forma sería hacer la media de los tiempos de lectura de los capítulos anteriores. Aunque esto sería mejor que nada, no podrías distinguir entre capítulos largos y cortos. Probablemente dedicarás menos tiempo a leer un capítulo corto que uno largo. Por lo que podrías considerar medir en minutos por página mejor que en minutos por capítulo.

Considera ahora, en la Tabla 6.1, los datos de los tiempos de la Estudiante Y dedicados a leer capítulos del libro. Los tiempos para leer capítulos varían desde 28 a 118 minutos. El cociente entre los tiempos mayores y menores es superior a 4. Cuando mides en minutos por página, el rango va desde los 2,33 minutos por página para el Capítulo 3 a los 7,38 minutos por página para el Capítulo 4. Esto es todavía un rango grande, ya que está entre uno y tres. Aunque habrá una variación considerable, intenta mejorar la estimación de los tiempos de lectura de los capítulos del libro, para ello, basa tu estimación en el tamaño del capítulo y en la velocidad media de lectura expresada en minutos por página. Para calcular la velocidad media de lectura, suma todos los tiempos de lectura y divide dicho valor por el número total de páginas de los capítulos:

$$\begin{aligned} \text{Velocidad media} &= (80 + 28 + 118 + 71 + 40) / (20 + 12 + 16 + 17 + \\ &\quad 12) = 337 / 77 \\ &= 4,38 \text{ minutos por página.} \end{aligned}$$

6.3

ALGUNAS PRECAUCIONES SOBRE LA UTILIZACIÓN DE LAS MEDIDAS DE TAMAÑO

Aunque utilizar las medidas de tamaño parece bastante sencillo, hay algunas complicaciones. Primero, algunos documentos son mucho más di-

Tabla 6.1 Tiempos de lectura de capítulos de la Estudiante Y.

Estudiante	Estudiante Y	Fecha	30/09/96
Profesor	Sr. Z	Clase	IP

Capítulo	Tiempo de lectura	Páginas	Minutos/página
1 y 2	80	20	4,00
3	28	12	2,33
4	110	16	7,38
5	71	17	4,18
6	40	12	3,33
Totales	337	77	
Medias	56,17	12,83	4,309

fíciles de leer que otros. Esto significa que tú deberías considerar el tipo de trabajo y no solo su tamaño.

También, hay una cuestión expuesta anteriormente. Supongamos, por ejemplo, que estás preparándote para un examen. Los tiempos para releer los capítulos estudiados anteriormente posiblemente serán inferiores a los tiempos iniciales de lectura. De forma similar, para releer un documento que has escrito, los tiempos de lectura podrían variar significativamente, dependiendo de la atención que se preste. Para una redacción inicial del borrador de un documento, puedes dedicarle de 15 a 20 minutos. Una hojeada rápida al documento completo, sin embargo, necesitaría un minuto por página o menos.

Cuestiones similares se presentan en la planificación del desarrollo de programas. Las productividades para diferentes clases de trabajo, tales como la reutilización de programas desarrollados previamente, la modificación de un programa existente o el desarrollo de nuevos programas, serán bastante diferentes. Para tratar estas cuestiones, deberías guardar de forma separada los registros de tamaño y tiempo para las distintas clases de trabajo que haces.

6.4

EL TAMAÑO DE UN PROGRAMA

Cuando estés estimando el tiempo requerido para codificar un programa, basa las estimaciones en los tiempos que anteriormente has dedicado a codificar programas parecidos. Como se muestra en la Tabla 6.2, los tiempos de la Estudiante Y para codificar programas oscilan en un rango de 69 a 158 minutos. Aunque este es un rango algo superior al doble, este aumentará en el futuro, cuando la Estudiante Y codifique programas más

Tabla 6.2 Tiempos de desarrollo de programas de la Estudiante Y.

Estudiante	Estudiante Y	Fecha	30/09/96
Profesor	Sr. Z	Clase	IP

Programa	Tiempo de desarrollo	LOC	Minutos/LOC
1	158	20	7,90
2	69	11	6,27
3	114	14	8,14
4	93	10	9,30
5	95	14	6,79
6	151	18	8,39
Totales	680	87	
Medias	110,0	14,5	7,82

grandes. De nuevo, es una buena idea basar las estimaciones de tiempo en el tamaño del programa.

La medida que utilizamos para el tamaño del programa, es la línea de texto del programa fuente. Así, si el programa tiene 16 líneas, el programa tiene 16 líneas de código (LOC, acrónimo de Lines Of Code). Al contar las LOC, se asume que no se cuentan las líneas en blanco o las líneas de comentarios. Aunque podrías escoger cualquier estándar coherente, en este libro contaremos las líneas de código sin contar las líneas en blanco o las líneas de comentarios. Así, el siguiente fragmento de programa en Ada. tiene 5 LOC.

Ejemplo 1

```
_ - comentario que describe la función del programa
  If (X_Media >= 100) then
    Tamaño := X_Media;
  else
    Tamaño := X_Media/2;
  end if;
```

De forma similar, si escribes ese mismo fragmento de programa sin comentarios y en un formato más comprimido, podrías tener 3 LOC:

Ejemplo 2

```
If(X_Media >= 100) then
  Tamaño := X_Media;
else Tamaño := X_Media/2; end if;
```

Aunque son programas idénticos y sus tiempos de desarrollo podrían ser los mismos, sus tamaños son diferentes con este método. Mientras mantengas la coherencia en la forma de codificar programas, estas variaciones de contabilizar no son importantes. Para asegurar que los tamaños contados sean coherentes, te sugiero que adoptes una forma normalizada para codificar los programas. Aunque sigas el formato propuesto por tu profesor, personalmente prefiero el formato más abierto que se muestra en el Ejemplo 1.

La medida de las LOC es aplicable a muchos lenguajes de programación. Por ejemplo, el mismo fragmento del programa en C++, sería el siguiente:

Ejemplo 3

```
// comentario que describe la función del programa
if (X_Media >= 100)
    Tamaño := X_Media;
else
    Tamaño := X_Media/2;
```

Cuando escribas de esta forma, este fragmento de código de C++ tendrá 4 LOC. El siguiente fragmento de código C++ , ligeramente superior tiene 12 LOC:

Ejemplo 4

```
while (n>0)
{
    push(n);
    cout << "Introducir un entero positivo. \n";
    cout << "Introducir 0 para terminar. \n";
    cin >> n;
}
// sacar de la pila
while (stack_pointer != NULL)
{
    cout.width(8);
    cout << pop ( );
}
```

Utilizando la técnica de las LOC para medir, los tamaños de los seis programas se muestran en la Tabla 6.2. El rango de minutos por línea de código (LOC) varía desde 6,27 del programa 2 a 9,3 para el programa 4.

Puesto que tus tiempos en minutos por LOC cambiarán bastante con la experiencia, deberías revisar la media de minutos por LOC y basar las estimaciones en las medias más recientes de los últimos 5-10 programas.

6.5

OTRAS MEDIDAS DEL TAMAÑO

Aunque no las utilicemos en este libro, hay muchas otras medidas del tamaño del software. Por ejemplo, el desarrollo industrial del software generalmente implica documentación, que se podría medir en páginas. Incluso para los programas, la medida de LOC no cubre todos los casos. Ejemplos de tipos de productos donde la LOC no es adecuada son los menús, los ficheros, las páginas de informes o las pantallas. A no ser que puedas concebir una adecuada medida del tamaño, deberías utilizar la contabilización de las unidades y las tasas vistas en el Capítulo 5.

También, si estás utilizando una herramienta de apoyo en el desarrollo de un programa que genera varias clases de pantallas, ventanas u otros elementos de programa estándar, la contabilización puede ser un poco más delicada. Aquí, es importante contar solamente las LOC que desarrolles y no las LOC generadas por las herramientas de apoyo de la programación.

Sin tener en cuenta las medidas utilizadas, el objetivo principal, es estimar el trabajo a desarrollar. Para esto, necesitas medidas de tamaño que relacionen el trabajo requerido, con el producto a desarrollar. Es decir, los desarrollos de productos que necesitan más tiempo deberían tener medidas de tamaño proporcionalmente mayores.¹

6.6

LA ESTIMACIÓN DEL TAMAÑO DEL PROGRAMA

Aunque ahora pareciera que puedes estimar el tiempo de codificar un programa, no es tan sencillo. Para leer los capítulos de un libro, podrías contar las páginas y utilizar la media histórica de minutos por página, para calcular el tiempo probable de lectura. Puedes clasificar los capítulos por dificultad de lectura y asignar una velocidad de minuto por página algo mayor para aquellos capítulos que parezcan más complejos o impliquen una materia con la que se esté menos familiarizado. Hojeando los nuevos capítulos, valoras su dificultad relativa y eliges una tasa (minutos/página) basada en experiencias anteriores.

¹ Para una discusión completa de las medidas y estimaciones del tamaño, véanse los Capítulos 4 y 5 de mi libro *A Discipline for Software Engineering* (Reading, MA: Addison-Wesley, 1995).

Para codificar programas, sin embargo, no es posible una contabilización de los mismos hasta que no los has desarrollado. Para estimar el tiempo de codificar un programa, tienes que estimar en primer lugar cuántas LOC requerirá y a partir de ahí, el número de minutos por LOC que probablemente necesitarás para desarrollarlo. Puedes entonces calcular el tiempo total estimado.

Aunque hay varios métodos para estimar los tamaños de los programas antes de desarrollarlos, todos los métodos de estimación de tamaño implican mucho juicio. Primero, examina los requisitos para los programas a desarrollar. Despues, clasifica los tamaños relativos de los nuevos programas entre los programas que has escrito ya. Finalmente, basándote en tu experiencia de dónde ubicas el nuevo programa dentro del histórico de rangos de tamaños, estima sus LOC.

Un ejemplo de este procedimiento se muestra en la Tabla 6.3. Es una lista de programas desarrollados anteriormente por la Estudiante Y, clasificados por tamaños. La lista muestra el tamaño del programa en LOC, el tiempo de desarrollo en minutos, las velocidades en minutos/LOC y una breve descripción de la función del programa. Examinando dichos datos en tus programas, y considerando que sabes bastante sobre el programa que vas a codificar, puedes juzgar dónde se ubicará este programa en la lista de tamaños. Esto te ayudará a estimar el rango de tamaño del nuevo programa. Basándose en los datos históricos de minutos por LOC, puedes estimar el tiempo para desarrollar el nuevo programa.

Por ejemplo, supongamos que la Estudiante Y planificó codificar un nuevo programa con un bucle while algo complicado. De la Tabla 6.3, estimaría que el tamaño es superior a las 14 LOC del programa 5 y probablemente menor a las 20 LOC del programa 1. A partir de lo anterior puede calcular la media de estos valores extremos, $(14+20)/2=17$ LOC.

Tabla 6.3 Rangos de tamaños de programas de la Estudiante Y

Estudiante	Estudiante Y	Fecha	30/09/96
Profesor	Sr. Z	Clase	IP

Programa	Tiempo	LOC	Minutos/LOC	Funciones
4	93	10	9,30	Bucle while sencillo
2	69	11	6,27	Sentencia case sencilla
3	114	14	8,14	Sentencia case grande
5	95	14	6,79	Bucle repetir-hasta medio
6	151	18	8,39	Lista enlazada sencilla
1	158	20	7,90	Cálculo pequeño

6.7

CÓMO HACER ESTIMACIONES DE TAMAÑOS MAYORES

Mientras la técnica mostrada en la Tabla 6.3 funciona razonablemente bien para programas pequeños parecidos a los que hayas codificado anteriormente, no funciona tan bien para nuevos tipos de programas o programas grandes. La razón es que incluso los programas más pequeños generalmente contienen una mezcla de funciones y procedimientos. Con programas más grandes, tendrás progresivamente más problemas al relacionar un nuevo programa con los programas desarrollados previamente.

Una solución a este problema es utilizar un formulario como el de la Tabla 6.4. Aquí, se pueden poner varios programas o funciones y procedimientos incluidos en programas. Con los datos de un cierto número de programas, podrías dividir la lista en categorías, como la Estudiante Y hizo en la Tabla 6.5. Ejemplos de otras clasificaciones útiles son: Texto, Control, Lógica, Pantalla e Impresión. El objetivo es construir un registro histórico de elementos previamente escritos junto con los datos de cuantas líneas de código contiene cada uno. Cuando consideres las funciones de un nuevo programa, puedes estimar el tamaño de cada función y sumar todas estas estimaciones de funciones para obtener la estimación total del programa.

Con programas pequeños, haz una lista para cada programa como la mostrada en la Tabla 6.5. Con programas de mayor tamaño, que contendrán probablemente múltiples funciones o procedimientos, enumerando cada función y procedimiento separadamente, puedes construir rápidamente una base histórica de datos de estimaciones. Si son muchos datos, es útil guardarlos en un formulario como el de la Tabla 6.4 para cada categoría de programa. Puedes tener un formulario para cálculos, otro para procedimientos que manejen texto, otro para control, etc. Para cada formulario, clasifica las funciones del programa por su tamaño e incluye el nombre o número del programa que la contiene.

Un ejemplo de cómo hacer una estimación de tamaño de esta manera, se muestra en la Tabla 6.5. Aquí, la Estudiante Y ha agrupado los datos de un número de programas en varias categorías. Puesto que estos programas eran todos bastante pequeños, puso solamente los programas completos. Para estimar el número de LOC que tendrá un nuevo programa, estimó cuantas LOC de cada tipo de función serían necesarias para el nuevo programa.

La Estudiante Y hizo esto, examinando en primer lugar los requisitos para el nuevo programa y estableciendo una estrategia general de cómo construirlo. Esperaba utilizar un bucle repetir-hasta y una sencilla sentencia case. También esperaba hacer bastantes cálculos sencillos. Haciendo estas estimaciones, sin embargo, no estaba completamente segura de qué tamaño tendrían cada una de las partes del programa, por ello estimó un

Tabla 6.4 Formulario para estimar el tamaño del programa.

Estudiante _____ Fecha _____
Profesor _____ Clase _____

Commentarios:

tamaño mínimo y uno máximo, a partir de ahí obtuvo el tamaño medio para cada función. Mientras utilizase los tamaños medios en sus estimaciones, el ejercicio de pensar en los tamaños máximo y mínimo le ayudaba a evitar hacer estimaciones que son o muy grandes o muy pequeñas.

Aunque este no es un método infalible de estimación de tamaños, es mucho mejor que estar adivinando. Conforme codifiques más programas, podrás añadirlo a la lista de rutinas. El aumento del número de datos históricos te ayudará a hacer progresivamente mejores programas.

No hay métodos que garanticen una buena estimación del tamaño. La estimación del tamaño es una habilidad. La clave para hacer buenas estimaciones de tamaño, es tener una cantidad sustancial de datos históricos, hacer muchas estimaciones de tamaño y comparar con regularidad tus resultados con las estimaciones.

6.8

CÓMO UTILIZAR MEDIDAS DE TAMAÑO EN EL CUADERNO DE TRABAJOS

Cuando utilizas las medidas de tamaño, en vez contar las unidades deberías guardar los datos del tamaño en el Cuaderno de Trabajos, tal y como

Tabla 6.5 Estimación del tamaño del programa de la Estudiante Y.

Estudiante Profesor	Estudiante Y Sr. Z			Fecha Clase	7/10/96 IP	
Programa	LOC	Func. anteriores	Funciones estimadas	Mín.	Media	Máx.
Bucles						
4	10	Bucle while sencillo				
5	14	Repetir-hasta sencillo	Repetir hasta	7	11	14
Case						
2	11	Sentencia case sencilla	Case	5	8	11
3	14	Sentencia case grande				
Datos						
6	18	Lista enlazada sencilla				
Calc.						
1	20	Cálculo pequeño	Calcular	10	15	20
Estimado				22	34	45

Comentarios: Este programa tiene una sentencia case sencilla, un bucle y un cálculo. Asumo que, como máximo, el tamaño se obtendrá sumando estos tamaños típicos, $11+14+20=45$ LOC. Para el valor mínimo, asumo que estas funciones podrían combinarse más efectivamente que cuando están como elementos separados. Esto nos da 22 LOC como el valor mínimo. 34 LOC es el punto medio entre los dos valores anteriores.

se muestra en las Tablas 6.6 y 6.7. Estos ejemplos utilizan los datos de tamaño de las Tablas 6.1 y 6.2. Aquí, en vez de anotar que uno o dos capítulos o programas fueron terminados, se ponen sus tamaños en la columna de unidades. Como se muestra en la Tabla 6.6, la Estudiante Y completó un programa de 20 LOC el 10/9, por ello puso un 20 en la columna U (unidades). El 11/9, leyó el Capítulo 3, puesto que dedicó 28 minutos a leer las 12 páginas, puso un 12 en la columna de unidades de la Tabla 6.6 y 28 en la columna de Δ Tiempo.

Puesto que tendrás datos de tamaño en el Cuaderno de Trabajos, puede parecer redundante ponerlos también en el Cuaderno de Registro de Tiempos. Si tu experiencia es como la mía, con frecuencia completarás el Cuaderno de Trabajos una o dos veces por semana. Entonces tendrás que buscar por datos de tamaño. En el momento que hagas el trabajo, sin em-

Tabla 6.6 El cuaderno de registro de tiempos con datos pequeños.

Estudiante _____ **Estudiante Y** _____ **Fecha** 9/9/96
Profesor _____ **Sr. Z** _____ **Clase** IP

bargo, no te llevará mucho tiempo escribir los datos de tamaño en el Cuaderno de Registro de Tiempos.

Tabla 6.7 El cuaderno de trabajo con datos pequeños.

Nombre: _____ Estudiante Y Fecha: _____ 9/9/96

En la Tabla 6.7, la Estudiante Y utilizó estos datos de tamaño para calcular velocidades del producto. Hizo esto utilizando las instrucciones del Cuaderno de Trabajos dadas en la Tabla 6.8. Por ejemplo, para el Trabajo 1 no hizo estimación de tamaño, dejando el espacio de Unidades Estimadas en blanco. El Tiempo Real permaneció en 158 minutos, pero las Unidades Reales son 20 LOC. Esto significa que las Unidades Hasta la Fecha son también 20 LOC y la velocidad Hasta la Fecha es $158/20=7,90$ minutos por LOC. Puesto que este es el primer programa, los valores Máximo y Mínimo también son 7,90.

Para el segundo programa, Trabajo 3, la Estudiante Y de nuevo no hizo estimaciones, dejó en blanco las Unidades Estimadas. El Tiempo Real para el programa 2 fue de 69 minutos y las Unidades Reales para el programa 2 fueron de 11 LOC. Esto da ahora para el Tiempo Hasta la Fecha un valor de 227 minutos y $20+11=31$ LOC para las Unidades Hasta la Fecha. La Estudiante Y obtuvo este valor buscando en las Unidades Hasta la Fecha la tarea de este tipo (Trabajo 1) más recientemente realizada, cuyo valor era 20 LOC y se lo añadió a las 11 LOC de las Unidades Reales. Ahora, con 31 LOC Hasta la Fecha y 227 de Tiempo Hasta la Fecha, el valor de la velocidad Hasta la Fecha es de $227/31 = 7,32$ minutos por LOC. La velocidad Máxima para el Trabajo 1 era 7,90, que es mayor que los 6,27 minutos por LOC para el Trabajo 3, por lo que la velocidad Máxima permanece en 7,90. La velocidad Mínima para el Trabajo 1 también era de 7,90, pero aquí, el valor de 6,27 minutos por LOC para el Trabajo 3 es menor que 7,90, por lo que el Mínimo es sustituido por 6,27, la velocidad del nuevo trabajo.

Sigue este mismo procedimiento para cada uno de los programas, hasta que hayas puesto todos los datos del Número de Trabajo para los programas codificados hasta la fecha. En el futuro, podrías estimar y escribir el valor de LOC para cada nuevo programa antes de escribirlo. Aunque esto lleva asociado un poco de trabajo, el procedimiento descrito en las Secciones 6.6 y 6.7 es relativamente fácil de seguir mientras tienes datos históricos. Para la lectura de textos, el procedimiento es idéntico, excepto que para determinar las Unidades Estimadas, debes contar el número de páginas que vas a leer del libro.

Utilizando estos datos, puedes ver cuánto tiempo has dedicado a codificar programas y a leer capítulos del libro. Podrías de igual forma incluir otras tareas, para las cuales tú tengas medidas de tamaño y tiempos.

RESUMEN

Este capítulo introduce las medidas de tamaño del producto y muestra cómo encajan en el proceso de planificación del producto.

Tabla 6.8 Instrucciones para el cuaderno de trabajos.

Propósito	Esta tabla se utiliza para controlar los trabajos de cada proyecto. Registra la información clave de cada proyecto. Un proyecto es cualquier actividad que deseas controlar, como desarrollar un programa o escribir un artículo.
Método	Cuando comiences un proyecto, escribe el número de trabajo en este cuaderno. Escribe números comenzando desde 1.
Cabecera	Introduce los siguientes datos: Tu nombre. La fecha de comienzo de este Cuaderno de Trabajos
Trabajo	Escribe el número de trabajo que has seleccionado.
Fecha	Escribe la fecha de inicio del trabajo.
Proceso	Escribe el tipo de tarea. Por ejemplo, para un trabajo técnico Escribe Artículo , para desarrollar un programa, utiliza Cod. , etc.
Tiempo Estimado	Escribe el tiempo total en minutos que se estimó para ese trabajo. Utiliza los valores de Velocidad Hasta la Fecha, Máximo y Mínimo como guía. Si estas velocidades no son razonables, utiliza tu experiencia.
Unidades Estimadas	Escribe las unidades estimadas para el trabajo acabado. Para el desarrollo de un programa, por ejemplo, estima el número de LOC que esperas que contenga el programa completo.
Tiempo Real	Escribe el tiempo total real para el trabajo realizado.
Unidades Reales	Escribe el número real de unidades totales. Para un programa, por ejemplo, podrías contar las LOC en el programa acabado.
Velocidad Real	Escribe el Tiempo Real dividido por las Unidades Reales.
Tiempo Hasta la Fecha	Localiza el trabajo más reciente de este tipo. Suma al valor de Tiempo Hasta la Fecha de este trabajo el Tiempo Real para el trabajo más reciente. Escribe este total en la casilla de Tiempo Hasta la Fecha para el nuevo trabajo.
Unidades Hasta la Fecha	Localiza el último trabajo realizado de este tipo. Suma las Unidades Hasta la Fecha de este trabajo a las Unidades Reales para el trabajo más reciente. Escribe este total en la casilla de Unidades Hasta la Fecha para el nuevo trabajo.
Velocidad Hasta la Fecha	Divide el tiempo Hasta la Fecha por las Unidades Hasta la Fecha para obtener los minutos por unidad para todos los trabajos terminados hasta la fecha.
Máx.	Escribe la velocidad máxima para todos los trabajos de cada tipo que hayan acabado.
Mín.	Escribe la velocidad mínima para todos los trabajos de cada tipo que hayan acabado.
Descripción	Escribe una descripción del trabajo que se ha hecho. Sé suficientemente claro para que el contenido del trabajo pueda ser fácilmente identificado.

Las medidas de tamaño se introducen para ayudar a estimar el tamaño del producto. El capítulo describe como determinar el tamaño de un programa en líneas de código (LOC). Otras medidas de tamaño, como los menús, los ficheros, las pantallas y las páginas de informes, son frecuentemente necesarias para proyectos de software a escala industrial.

El primer paso para planificar un producto es estimar su tamaño. Para hacer una estimación exacta del tamaño, utiliza datos históricos. Es útil dividir los datos históricos del tamaño en categorías funcionales. Puedes estimar cuántas líneas de cada categoría necesitarás en el nuevo programa. Conforme acumules datos históricos, probablemente harás estimaciones más exactas. El Cuaderno de Trabajos proporciona una forma adecuada para registrar grandes volúmenes de datos históricos de tamaño y velocidad.

EJERCICIO 6

Revisa las tareas finalizadas en los ejercicios hechos hasta la fecha y cuenta sus tamaños en LOC y páginas. Presenta estos datos en el formato de la Tabla 6.1 para los ejercicios de **los** capítulos y en el de la Tabla **6.3** para los programas. También, entrega actualizados, el Resumen Semanal de Actividades y el Cuaderno de Trabajos, con los datos de velocidades expresadas en minutos por LOC y minutos por página. Si no tienes copias de todos los programas acabados, incluye datos de al menos **los 3** Últimos. También, presenta una copia de algunas páginas del Cuaderno de Registro de Tiempos, del Cuaderno de Trabajo y del Resumen Semanal de Actividades que no hayas entregado anteriormente. A partir de ahora, presenta en donde proceda, todos los cuadernos y resúmenes con tasas en minutos por LOC y minutos por página.

CAPÍTULO 7

La gestión del tiempo

Este capítulo describe las estimaciones de tiempo y muestra cómo usarlas. Explica cómo hacer una estimación de tiempo y sugiere algunas técnicas para satisfacerlo. El ejercicio del capítulo consistirá en hacer una estimación de tiempo.

7.1 ELEMENTOS DE LA GESTIÓN DEL TIEMPO

Hasta ahora, a lo largo de este curso, durante varias semanas hemos medido a qué dedicamos nuestro tiempo. Disponemos de una modesta cantidad de datos y una buena idea de cuánto tiempo dedicamos a realizar diversas tareas. Este conocimiento no ha sido difícil de conseguir, y ahora comprobamos cómo te ayudan estos datos a hacer planes futuros. En este capítulo, los utilizaremos para gestionar tu tiempo de la siguiente forma:

- 1.** Decidir cómo quieres utilizar tu tiempo.
- 2** Hacer una estimación de tiempo.
- 3.** Controlar la forma de utilizar el tiempo frente a lo estimado.
- 4.** Decidir qué cambios hacer para llevar tus acciones en concordancia con lo estimado.

Estos temas se discuten en las secciones siguientes.

7.2

LA CLASIFICACIÓN DE LAS ACTIVIDADES

Revisa las categorías de tiempo para ver si cubren todas tus actividades principales. Si no es así, añade más, revisándolas. Puedes encontrar, por ejemplo, que dedicas un tiempo significativo a analizar y utilizar los datos que has reunido en este curso. En este caso, te conviene añadir una categoría de análisis de datos al Resumen Semanal de Actividades.

Aunque tengas ahora una buena idea del tiempo que dedicas semanalmente a las tareas, tendrás muchos datos sobre tareas que surgen de forma ocasional a lo largo de un mes, semestre o año, como estudiar para los exámenes o trabajar en proyectos trimestrales. Naturalmente, no puedes reunir datos de estas tareas hasta que las hagas, pero intenta identificar estas categorías ahora, para que puedas hacer algunas previsiones de ellas en tus planes.

7.3

LA RECOGIDA DE DATOS SOBRE EL TIEMPO DEDICADO A CADA ACTIVIDAD

El Resumen Semanal de Actividades muestra los tiempos medio, máximo y mínimo que se dedican a cada actividad durante la semana. Es una buena idea examinar estas categorías ahora, para ver si son muy generales o muy detalladas. Una distribución desigual puede hacer, que una categoría tenga un 50% o más de tiempo y las otras un 5% o menos.

Para gestionar el tiempo, necesitas centrarte en esas pocas categorías que consumen la mayor parte de tu tiempo. Necesitarás saber con algún detalle qué haces. Si dedicas el 25% de tu tiempo a categorías etiquetadas como Otras, puedes dividirlas en un par de actividades mejor definidas. Sin datos más precisos de estas Otras actividades, tendrás problemas en controlar cuánto tiempo dedicas a ellas.

7.4

CÓMO EVALUAR TU DISTRIBUCIÓN DEL TIEMPO

Ahora que puedes saber cómo utilizas tu tiempo, pregúntate a ti mismo si estás utilizando el tiempo de la forma que quieras. Decide qué actividades son más importantes y considera si estás dedicándole el tiempo suficiente. ¿A algunas tareas, le dedicas más tiempo que a otras que son más importantes? ¿Estás dejando suficiente tiempo para leer el libro de texto? ¿Haces el trabajo? Y, jcuáles son tus compromisos personales? ¿Comienzas los ejercicios a tiempo para acabarlos, o los terminas en el último momento? No hay guías generales sobre cómo utilizar tu tiempo.

Esta es una decisión muy personal que debes equilibrar entre el trabajo académico, las tareas, el descanso y la vida social. Algunos de estos componentes son cuestiones personales que implican complejas elecciones, particularmente si tienes un trabajo y responsabilidades familiares.

7.5

CÓMO HACER UNA ESTIMACIÓN DE TIEMPO

La estimación de tiempo es tu plan de cómo utilizar el tiempo. Comenzando por los datos de cómo has utilizado anteriormente el tiempo, puedes asignar cantidades de tiempo que probablemente utilizarás en cada categoría en el futuro. La estimación de tiempo preliminar para la Estudiante Y se muestra en la Tabla 7.1.

Tabla 7.1 Ejemplo de presupuesto semanal de tiempo.

Estudiante Estudiante Y
Profesor Sr. Z Fecha 23/9/96
Clase IP

Actividad	Minutos estimados	Minutos reales
Asistir a clase	150	
Escribir programas	360	
Leer Texto	100	
Preparar exámenes	120	
Otros	30	
Total	040	

La Tabla 7.2 es una copia del Resumen Semanal de Actividades de la Estudiante Y de la Tabla 4.4. Observando la línea 19, puedes comparar la estimación de tiempo en la Tabla 7.1 con la forma de utilizarlo realmente. Esto nos conduce a las siguientes conclusiones:

- El tiempo total de la Estudiante Y en este curso tiene una media de 742 minutos por semana hasta la fecha. Es decir 12,37 horas. Con la nueva estimación, ella planifica dedicarle 840 minutos, o sea 14 horas por semana, lo que supone un tiempo adicional de 1,5 horas. Este es un cambio demasiado grande ya que puede tener problemas para encontrar este tiempo adicional cada semana.
- La Estudiante Y está estimando la misma cantidad de tiempo para asistir a clase, tal y como venía haciéndolo hasta la fecha, es un plan razonable.
- Planifica dedicar 20 minutos más de tiempo cada semana a codificar programas. Esto, probablemente, también es razonable.

Tabla 7.2 Resumen semanal de actividades de la Estudiante Y.

	Nombre	Estudiante Y					Fecha	23/9/96	
1	Tarea	Clase	Codificar	Prep.	Leer	Rev.		Otros	Total
2	Fecha		Prog.	Exam.	Texto				
3	D 15/9								
4	L	50	93		80				223
5	M		95						95
6	Mi	50			71				121
7	J		77						77
8	V	50	74		40				164
9	S				33				33
10	Totales	150	339		224				713

11 Tiempos y medias del período Número de semanas (número anterior +1): 2

12 Resumen de las semanas anteriores

13	Total	150	341	134	146				771
14	Med.	150	341	134	146				771
15	Máx.	150	341	134	146				771
16	Mín.	150	341	134	146				771

17 Resumen incluyendo la última semana

18	Total	300	680	134	370				1484
19	Med.	150	340	67	185				742
20	Máx.	150	341	134	224				771
21	Mín.	150	339	134	146				713

- Planifica dedicarle, aproximadamente, el mismo tiempo que el empleado hasta ahora para leer libros. Esto parece razonable.
- Los grandes incrementos están en las categorías de preparar exámenes y Otros. Estas dos suman un total de 150 minutos por semana, o 2,5 horas. Este gran cambio, probablemente, no es realista. Aunque, puede tener sentido para la Estudiante Y, aumentar su tiempo total a un valor próximo al máximo de los 771 minutos dedicados hasta ahora, esperar hacer mucho más podría no ser realista.

La clave para gestionar el tiempo es reequilibrar gradualmente la forma de utilizarlo. Así, no puedes esperar dedicar más tiempo a algunas tareas en el futuro, a menos que identifiques otras áreas que puedas acortar, sino esto es ilusorio. Un paso importante es asegurarte que utilizas tu tiempo con más

efectividad. A menudo, las personas gastan mucho tiempo decidiendo qué van a hacer próximamente. Estableciendo planes personales y estimando el tiempo, podrán *saber* qué hacer a continuación. Asombrosamente, esto mejorará de inmediato su eficiencia en el trabajo.

Puedes necesitar más tiempo para hacer todas las cosas que deseas hacer. De nuevo, sin embargo, necesitas ser realista. Normalmente, puedes encontrar algún tiempo extra en una situación de crisis, pero no quieras vivir perpetuamente en crisis. Así, dedica cantidades razonables de tiempo al cuidado de tu salud, a la familia, a los amigos, a las aficiones y a las comidas. Aunque podrás ser capaz de hacer algunos ajustes en estas áreas, deberías hacerlo de forma gradual. Te sugiero que establezcas un ritmo sostenido de trabajo que puedas llevar de forma continua durante el curso académico. Puedes ser capaz de recortar profundamente una o más áreas en una crisis, pero no puedes hacerlo durante mucho tiempo. Si debes dedicar más tiempo a alguna actividad, necesitas tomar este tiempo de otra parte. Entonces haz tu estimación de tiempo de acuerdo con esto.

7.6

CÓMO ENCONTRAR MÁS TIEMPO

Después de haber revisado la estimación de tiempo, puedes necesitar aumentar la cantidad total de tiempo. ¿Comó puedes hacer esto? Tienes varias opciones.

Primero, si tu agenda no está muy ocupada, serás capaz de encontrar un poco de tiempo extra, pero desdichadamente, pocas personas están bendecidas con el tiempo libre.

Es más probable que estés super comprometido. En este caso, haz un amplio estudio de todos tus compromisos. Despues revisa el tiempo que utilizas tanto en las clases, como en las principales áreas de trabajo, así como en las actividades de ocio.

El problema es que el día solamente tiene **24** horas y no hay nada que puedas hacer para aumentarlas. Si necesitas dedicar más tiempo a alguna actividad, debes tomar ese tiempo de otras actividades. Así, hasta que no sepas cuanto tiempo dedicas a cada categoría de actividades, será difícil hacer ajustes realistas.

7.7

CÓMO ESTABLECER REGLAS BÁSICAS

Ahora que has decidido cómo quieres utilizar el tiempo, necesitas realmente utilizarlo de esa forma. Esta regla tan sencilla lleva más esfuerzo del que puedas prever. La razón es que las estimaciones de tiempo son,

Tabla 7.3 Estimación semanal de actividades (*Continuación*).**Estimación Semana #2**

Tarea									Total
Fecha									
D									
L									
M									
Mi									
J									
V									
S									
Totales									

Estimación Semana #3

Tarea									Total
Fecha									
D									
L									
M									
Mi									
J									
V									
S									
Totales									

a cada actividad. Poner excedentes, te permitirá gestionar errores ocasionales de planificación y eventos imprevistos.

Una estimación de tiempo más realista para la Estudiante Y se puede ver en la Tabla 7.4.

7.8 CÓMO PRIORIZAR TU TIEMPO

Un paso esencial en la gestión del tiempo es establecer prioridades. Algunas veces están firmemente establecidas, como asistir a clase o cuándo trabajas. Podrías llamar a esto, tiempos fijos. Cualquier otra cosa, es el tiempo va-

riable: actividades que haces cuando puedes encontrar tiempo. Hay, sin embargo, dos tipos de actividades variables: las exigidas y las discrecionales. Las actividades exigidas incluyen tareas como, hacer en casa los trabajos asignados en clase, leer un libro o preparar los exámenes. Aunque son exigidas, son variables, porque las haces si encuentras tiempo, y dedicas a las mismas, cantidades variables de tiempo cada semana. Las actividades discretionales son todas las otras cosas que haces: comer, dormir, vida social, hacer deporte, ver deporte en la televisión y otros entretenimientos.

Cuando haces una estimación global de tiempo, es útil determinar de forma exacta cuánto tiempo has dedicado a cada categoría. Programar los

Tabla 7.4 Ejemplo de estimación semanal de actividades.

Nombre Estudiante Y _____ Fecha 23/9/96 _____

Estimación Semana #1 Estimación para una semana normal

Tarea	Clase	Codificar	Prep.	Leer	Rev.			Otros	Total
Fecha		Prog.	Exam.	Libros					
D									
L	50			40					90
M		120		40					160
MI	50			40					90
J		120		40					160
V	50			40					90
S		120							120
Totale	150	360		200					710

Tarea	Clase	Codificar	Prep.	Leer	Rev.			Otros	Total
Fecha		Prog.	Exam.	Libros					
D									
L	50		40						90
M		120	40						160
MI	50		40						90
J		120	40						160
V	50		120						170
S			150						150
Totales	150	240	430						820

Tabla 7.4 Ejemplo de estimación semanal de actividades (*Continuación*).**Estimación Semana #3** Semana del examen

Tarea	Clase	Codificar	Prep.	Leer	Rev.			Otros	Total
Fecha		Prog.	Exam.	Libros					
D									
L			300						300
M			450						450
Mi	170		150						320
J	50								50
V									
S									
Totals	220		900						1120

tiempos fijos no es problema, el problema más común es, asignar el tiempo variable. Por ejemplo, si no tienes suficiente tiempo para hacer los trabajos asignados cuando debas, el Único sitio para obtener tiempo adicional es de tus actividades discrecionales. Esto sugiere que eches un vistazo a los tiempos fijos, exigidos y discrecionales para ver donde puedes hacer los ajustes.

Una herramienta para examinar la distribución personal del tiempo es el resumen global de tiempos que se muestra en la Tabla 7.5. Para completar dicha tabla, tendrás que registrar, aproximadamente, cuanto tiempo dedicas normalmente a cada actividad. Aunque, podrías utilizar el Cuaderno de Registro de Tiempos, no se necesitará, en una primera aproximación este grado de detalle. Para empezar, es adecuado anotar al final de cada día, a qué has dedicado tu tiempo en cada categoría. Así te aseguras que los tiempos de cada día suman 24 horas y el tiempo total de la semana suma 168 horas o sea 10.080 minutos, tus totales probablemente estarán bien. Si no eres capaz de hacer un resumen adecuado de esta forma, entonces puedes utilizar el Cuaderno de Registro de Tiempos.

Con un resumen total como el que se muestra en la Tabla 7.5, tienes los datos básicos para decidir de donde coger el tiempo adicional que necesitas. De nuevo, sin embargo, necesitas ser realista. Reducir las horas de sueño o de comer, por ejemplo, puede dañar tu salud y disminuir la eficiencia del trabajo. Saltarse comidas y trasnochar puede ser una forma de solucionar problemas a corto plazo, pero generalmente, no es una buena idea planificar importantes reducciones en estas áreas.

Siendo realista, surge otro aspecto: tu buena voluntad para trabajar realmente en el plan establecido. Una planificación sin tiempo para la vida social, la relajación o los ejercicios físicos, puede llegar a resultar un fastidio. Aunque puedas reducir el tiempo dedicado al relax y a la diver-

Tabla 7.5 Resumen global de los tiempos semanales.

Nombre	Estudiante Y				Fecha	23/9/96	
	Z						
Actividad	Informát.	Física	Matem.	Inglés	Comer/ Descanso	Otros	Total
Fija							
Clases	150	150	100	100			500
Exigida							
Trabajo casa	360	240	240	360			1200
Leer libro	240	240	180	60			720
Discrecional							
Comer					1260		1260
Dormir					3150		3150
Deportes						600	600
Entretenimiento						360	360
Relajación						2290	2290
Totales	750	630	520	520	4410	3250	10.080

sión, es necesaria una cierta cantidad de ocio. Es fácil llegar a estar cansado y depresivo cuando te concentras exclusivamente en el trabajo. Así como rendirse a unas patatas fritas, al chocolate o a un helado, puede causarte que frustres una dieta, esto te podría conducir a rebelarte contra la idea global de la gestión del tiempo. Aunque es importante estimar y revisar el tiempo, asegúrate de que tu estimación es algo que estás dispuesto a llevarlo a cabo con buena voluntad.

Conforme vayas controlando el tiempo, compara el tiempo real dedicado frente al tiempo estimado. Si puedes gestionar el tiempo consistentemente, de acuerdo a tu estimación, no necesitas hacer muchos cambios. Sin embargo, lo más probable es que tu plan inicial necesite ajustes. No te sientas mal si tienes que cambiar las estimaciones básicas. Puesto que las actividades cambiarán de semana en semana, necesitarás hacer revisiones ocasionalmente. Recuerda, sin embargo, anotar las nuevas estimaciones.

7.9

LA GESTIÓN DEL TIEMPO ESTIMADO

Para establecer las reglas básicas de la gestión del tiempo, utiliza la aproximación que se muestra en la Tabla 7.6, anota para cada día **los** tiempos

de cada categoría. Si quieres utilizar dos períodos de tiempo para una categoría en un día, utiliza dos columnas o haz más grande los espacios de este formulario. El formato no es lo importante, céntrate, más bien, en establecer un conjunto de reglas explícito y claro.

Tabla 7.6 Ejemplo de estimación y gestión semanal del tiempo.

Nombre Estudiante Y Fecha 23/9/96

Estimación Semana #1 Estimación para una semana normal

Tarea	Clase	Codificar	Prep.	Leer	Rev.			Otros	Total
Fecha		Prog.	Exam.	Libros					
D									
L	9:00–9:50			10:20–11:00					90
M		8:30–10:30		10:20–11:00					160
MI	9:00–9:50			10:20–11:00					90
J		8:30–10:30		10:20–11:00					160
V	9:00–9:50			10:20–11:00					90
S		8:30–10:30		10:20–11:00					160
Totales	150	360		240					750

Tu capacidad para trabajar de acuerdo al tiempo estimado dependerá en gran parte de la disciplina personal, pero también dependerá del número y prioridad de las cosas que estás intentando hacer. Los asuntos inesperados, son cosas naturales y normales de la vida, especialmente en la ingeniería del software. Los problemas alterarán tus planes periódicamente y tendrás que hacer ajustes. Por ejemplo, para cumplir una fecha importante, puedes tener que trabajar hasta muy avanzada la noche o aplazar las actividades planificadas de tipo familiar, ocio o social.

Puedes encontrar, que la primera vez que utilizas la estimación de tiempo, no es muy útil. Esto es normal, no renuncies al proceso de estimar el tiempo porque no funcione la primera vez. En su lugar, piensa qué es lo que ha sucedido. ¿Fue algún suceso inusual, que probablemente no volverá a suceder? o ¿fue el tiempo inesperadamente gastado en alguna actividad normal? Si fue una emergencia verdadera, no tendrás necesidad de hacer cambios radicales en la estimación. Intenta utilizarlo otra sema-

na, y vuelve a examinar de nuevo los resultados. Si, por el contrario, la estimación fue alterada por algún suceso normal, considera ajustar la estimación para adelantarte a dichos sucesos en el futuro.

7.10

SUGERENCIAS PARA LA GESTIÓN DEL TIEMPO VARIABLE

Cuando establezcas las reglas básicas para la gestión del tiempo variable, considera las siguientes cuestiones:

- *¿Qué actividades son de máxima prioridad?* Intenta hacer las tareas más importantes primero. Es natural aplazar las tareas difíciles o poco interesantes, pero nunca se volverán fáciles. Si tú mismo estás retrasando trabajo importante, detente y piensa un poco lo que estás haciendo. Verás que cuando aplazas tareas importantes, inconscientemente te preocupas por ellas. A menudo, en efecto, decidir hacerlas de forma correcta, será más eficiente y te proporcionará un sentido útil de logro. Recuerda también, que una vez que has iniciado las tareas temidas, raramente volverán a ser tan difíciles como creías.
- *¿Hay algunas tareas que se deberían realizar en momentos específicos?* Por ejemplo, estudiar para un examen, prepararte para una sesión de laboratorio o reunirte con tu tutor de la Facultad. Asigna tiempos específicos a estas demandas en tu presupuesto de tiempo.
- *¿Hay actividades que quieres hacer tan pronto como tengas tiempo?* En tu cuaderno de ingeniería, guarda una lista de cosas que necesitas hacer lo antes posible, como codificar los programas asignados o leer el siguiente capítulo del libro. Entonces, puedes comprobarlas cuando las termines.

Para gestionar diariamente tu tiempo, necesitas llevar las estimaciones de la gestión del tiempo contigo en todo momento. Una buena idea para hacer esto, es llevártelas en tu cuaderno de ingeniería. Insértalos con un clip en la parte delantera o trasera del cuaderno de ingeniería, o grápalos. Cuando actualices el presupuesto, toma el más reciente y guarda las copias antiguas. Comprueba las versiones anteriores antes de hacer un cambio, pues te ayudarán a evitar hacer correcciones. Para compensar haber sobreestimado una tarea estimada, por ejemplo, probablemente la subestimarás la próxima vez. Si examinas las estimaciones anteriores, te mostrarán la tendencia y te ayudarán a escoger un valor medio más documentado. Una regla para sobrevivir en la ingeniería del software es: no tires los planes, datos o programas viejos. Asombrosamente, los necesitarás después.

7.11

TU OBJETIVO EN LA GESTIÓN DEL TIEMPO

Después de haber gestionado tu tiempo de esta forma durante unas pocas semanas, considera simplificar la recolección de datos, mediante la consolidación de varias categorías de tiempo en unas pocas. Tu objetivo en este punto, es conseguir una visión global de tu distribución de tiempo, antes que conocer los detalles. Por ejemplo, después de revisar mis tiempos de varios años, yo he reducido mi tiempo de trabajo solamente a dos categorías: proyectos de empresa y proyectos personales. Puesto que yo planifico cada proyecto separadamente, un análisis de un periodo detallado, no añade más información útil. Para mí, la cuestión más importante, es saber cuánto tiempo puedo dedicar a cada una de mis dos categorías de proyecto cada semana o mes.

Recuerda que reunir los datos de tiempos te ayudará a ti mismo a gestionar el tiempo. Si los datos que reúnes no son útiles, reconsidera la forma de agruparlos. Haz esto, solo después de tener práctica en estimar el tiempo. Aún así, si por alguna razón la distribución de tu tiempo cambia significativamente, reúne más datos hasta que entiendas cómo utilizas tu tiempo realmente.

RESUMEN

Para gestionar bien tu tiempo analiza tus propios datos históricos de tiempos. Establece una estimación para utilizar el tiempo y registra tu tiempo real frente al estimado. Para hacer una estimación de tiempo decide cómo quieres utilizar el tiempo. Haz una programación que refleje tu elección y que muestre los tiempos cada día; puedes necesitar diferentes estimaciones para distintas semanas.

Las reglas básicas para estimar el tiempo pueden ser útiles: identifica tus compromisos fijos y variables. Divide tu tiempo variable en tareas que son exigidas y aquellas que son discretionales. Analiza como divides ahora tu tiempo en estas categorías. Recuerda que tu tiempo total es fijo: si necesitas más tiempo para algunas actividades, debes dedicar menos tiempo a otras.

Finalmente, revisa el rendimiento frente al tiempo estimado: continúa reuniendo datos de tiempos. Revisa el tiempo estimado frente a tu experiencia real. Revisa la estimación basándote en tus necesidades y experiencias. Haz los cambios de forma gradual. Cuando cambies tu estimación de tiempo, guarda las versiones anteriores.

EJERCICIO 7

Utilizando los datos de la forma que tienes de gastar el tiempo, establece una estimación de tu tiempo para semanas normales y para alguna especial que prevés durante el resto del semestre. Para los casos donde tu estimación difiera sustancialmente de los tiempos medios, explica brevemente porque haces las distintas elecciones. Utiliza los formatos de estimación mostrados en las Tablas 7.4 y 7.6. Entrega varias copias de estas estimaciones con el trabajo asignado para casa. Observa que la estimación de tiempo es un ejercicio opcional que no necesitas completar a no ser que te lo pida tu profesor.

Entrega las copias de las páginas del Cuaderno de Registro de Tiempos y del Resumen Semanal de Actividades que no hayas entregado todavía.

CAPITULO 8

La gestión de los compromisos

Este capítulo se centra en los compromisos, discute qué son, por qué son importantes y cómo se gestionan. En el ejercicio revisarás y anotarás tus compromisos.

8.1 DEFINICIÓN DE COMPROMISO

Estar comprometido, es un estado mental. Por cualquier razón, te has comprometido a hacer algo, y sientes que deberías hacerlo. Un compromiso, sin embargo, es algo más que hacer lo que deseas hacer, hay algo que alguien espera que hagas. En efecto, esta es la cuestión clave en los compromisos: ¿quién es la persona con la que te has comprometido? En sentido legal o contractual, estás comprometido con alguien: tu profesor, tu director, tu jefe. Más importantes, sin embargo, son los compromisos íntimos que haces contigo mismo.

El principal problema con muchas programaciones y planes de software, es que la dirección las ve como compromisos contractuales, pero los ingenieros del software no las ven como compromisos personales. La diferencia, como veremos, está en gran parte, en cómo se hacen los com-

promisos. En este capítulo, aprenderemos cómo hacer compromisos contractuales que sean también compromisos personales.

Con un compromiso contractual, dos o más personas deben ponerse de acuerdo antes sobre la acción a realizar, esto es un compromiso. Por ejemplo, el Sr. A y la Sra. B llegan al acuerdo de que el Sr. A proporcionará algún producto o realizará alguna tarea para la Sra. B. Un ejemplo, es tu compromiso con tu profesor de hacer el ejercicio asignado para este curso. Otro ejemplo, podría ser tu contrato de escribir un programa para un cliente.

Cuando el Sr. A se compromete, se pone de acuerdo con la Sra. B, para realizar una tarea específica en un tiempo determinado y por una recompensa o compensación. Esto indica que hay dos elementos adicionales en los compromisos. Además de comprometerte en la tarea, las partes también se ponen de acuerdo sobre el tiempo en que va a ser hecha y en el pago u otra retribución que recibirá el Sr. A. Un ejemplo podría ser la obligación del cliente de pagarte por el desarrollo e instalación de algún software.

Una característica de los compromisos personales es que son voluntarios. Supongamos, por ejemplo, que tu cliente necesita el programa antes y te dice que lo acabes dos semanas antes de lo inicialmente establecido. Él, nunca preguntó si podías realizar la proeza, y tú, por tu parte, no te comprometiste. Vosotros comentáis la nueva fecha. Aunque pudieras comprometerte con la nueva fecha, probablemente no querrás sentirte personalmente comprometido a hacerlo.

Para estar verdaderamente comprometido, debes de considerar atentamente las alternativas y decidir que esto es algo que tú puedes hacer y harás. Cuando te ordenan que debes hacer algo, no se tratará de un compromiso personal. En efecto, cuando a las personas se les ordena hacer cosas, a menudo, se sienten amenazadas y enfadadas. Se resisten a la persona que dicta las ordenes y puede que quieran responder contra ellos. Una forma de respuesta, naturalmente, sería no hacer la acción demandada. Dicha reacción en los negocios puede parecer infantil, pero muchas personas responden inconscientemente de esta forma.

El verdadero acuerdo es la característica más importante de un compromiso personal. Las partes deben ponerse de acuerdo sobre qué hacer, cuando se terminará y qué es lo que se dará a cambio.

Un verdadero compromiso, es tanto personal como contractual y requiere un acuerdo voluntario y explícito entre dos o más partes sobre:

- Qué se hará.
- Los criterios para determinar que está hecho.
- Quién lo hará.

- Cuándo se hará.
- La compensación u otra retribución que se dará a cambio.
- Y quién proporcionará la compensación o retribución.

8.2

RESPONSABILIDAD PARA HACER COMPROMISOS

Además de las características ya descritas, los compromisos deberían hacerse responsablemente y gestionarse de manera adecuada. Puedes asegurarte de que tus compromisos son responsables y están bien gestionados de la siguiente forma:

Analiza el trabajo antes de aceptar el compromiso. Ambas partes deben establecer el compromiso de buena fe. Tú estás personalmente comprometido y realmente pretendes hacer el trabajo y la otra parte está dispuesta a proporcionarte la adecuada remuneración a cambio. La cuestión, sin embargo, es el grado en que podéis asegurar que podéis cumplir el compromiso. Por ejemplo, ¿has examinado el trabajo con suficiente detalle para saber que lo puedes hacer? De forma similar, ¿la otra parte tiene la capacidad de remunerarte? Con frecuencia, los compromisos software están basados en algo más que esperanzas. Cuando ambas partes pretenden verdaderamente llevarlo a cabo, con buenas intenciones no se proporciona una base razonable para un buen acuerdo.

Apoyar el compromiso con un plan. Para un trabajo de cualquier tamaño, la forma de hacerse responsable de un compromiso, es hacer en primer lugar un plan para el trabajo. La planificación implica algún esfuerzo, pero no es necesario que sea muy grande. En efecto, si has tenido experiencia en hacer planes formales, puedes normalmente completarlos rápidamente.

Documentar el compromiso. Aunque esto puede parecer obvio, no lo es. Hay un malentendido común de que las personas honestas solo necesitarán unas pocas palabras y un apretón de manos. Pero las palabras, a menudo, son mal interpretadas. Después de que dos personas se comprometen oralmente, tienen, a menudo, problemas de comprometerse por escrito. Esto significa que sus compromisos originales eran superficiales e irreales. El segundo problema tiene que ver sobre qué harán las dos partes si surgen problemas. Esta, en efecto, es la principal razón de muchos contratos. No necesitas un contrato cuando las cosas funcionan de acuerdo con lo planificado, lo necesitas cuando hay problemas.

Si eres incapaz de cumplir el compromiso, díselo inmediatamente a la otra parte e intenta minimizar el impacto sobre esa persona. Cuando hayas aprendido a gestionar tus compromisos, casi siempre los cumplirás. Desdichadamente, aun con los mejores planes, el trabajo ocasional-

mente será más complejo de lo que esperabas o puede surgir algo imprevisible.

8.3

EJEMPLO DE UN COMPROMISO

La Estudiante Y se ha comprometido a hacer 10 horas de trabajo a tiempo parcial cada semana en la Oficina de Administración de la Universidad. Los pasos necesarios para establecer adecuadamente el compromiso para este caso, se muestran a continuación.

1. La Estudiante Y se reúne con el jefe de la Oficina de Administración, que le explica el trabajo que deberá hacer. Esta es la fase de requisitos del proceso de compromiso, en la cual la Estudiante Y se entera de qué clase de trabajo se espera que haga. Para un trabajo de oficina, el producto de esta fase debería ser una lista de tareas a realizar, posiblemente una descripción del trabajo.
2. La Estudiante Y analiza las tareas y concluye que está dispuesta a hacer ese trabajo y es capaz de llevarlo a cabo.

Respondida la primera cuestión: ¿puedo hacer el trabajo?, la siguiente cuestión es: ¿puedo hacerlo en el tiempo y con los recursos requeridos? Para responder a esta cuestión, la Estudiante Y hace lo siguiente:

3. Examina **sus** compromisos personales de tiempo y concluye que puede encontrar 10 horas (*o sea* 600 minutos) durante cada semana para hacer el trabajo. Como puedes observar, comparando la Tabla 8.1 con la Tabla 7.5, obtiene este tiempo, reduciendo su tiempo total de descanso en 600 minutos.
4. Revisa **sus** propios compromisos, mostrados en la Tabla 8.2, y concluye que la franja horaria que necesita para el trabajo, de 4:00 a 6:00 de la tarde de lunes a viernes, está disponible.
5. Ahora que está segura de que puede hacer el trabajo, discute, la cantidad de dinero a percibir y la fecha de incorporación al trabajo, con el jefe de la Oficina de Administración.
6. Con todas estas cuestiones establecidas, la Estudiante Y y el jefe se comprometen en todas las cuestiones discutidas y el supervisor se compromete a proporcionarle una carta que resuma los puntos clave.

Este es un compromiso responsable. La Estudiante Y se tomó su tiempo para entender el trabajo y su capacidad para hacerlo. Se aseguró que podía hacer el trabajo y que tendría el tiempo disponible. También se llegó a un acuerdo con el jefe sobre la cantidad de dinero a percibir y otras condiciones del trabajo.

Tabla 8.1 Resumen semanal de tiempos semanales de la Estudiante Y.

Nombre	Estudiante Y					Fecha	3019196	
Profesor	Sr. Z					Clase	IP	
Actividad	Informát.	Física	Matem.	Inglés	Comer/Descanso	Otros	Total	
Fija								
Clases	150	150	100	100				500
Trabajo							600	600
Exigida								
Trabajo casa	360	240	240	360				1200
Leer libro	240	240	180	60				720
Discrecional								
Comer					1260			1260
Dormir					3150			3150
Deportes							600	600
Entretenimiento							360	360
Relajación							1690	1690
Totales	750	630	520	520	4410	3250	10.080	

8.4 UN EJEMPLO EN LA INDUSTRIA

A un ingeniero de una de las principales compañías de automóviles, le asignaron un trabajo de software urgente. Antes, la dirección le había dado simplemente la fecha junto con el trabajo asignado. Pero puesto que el ingeniero y su equipo habían terminado el curso de PSP, pidió tiempo para echarle un vistazo al problema y volver a la mañana siguiente con un plan.

El ingeniero estimó el tamaño del trabajo y utilizó sus datos del PSP para determinar cuánto tiempo le llevaría el trabajo. Volvió a hablar con su director y le explicó el plan y por qué creía que era realista. Su director estuvo de acuerdo y el ingeniero acabó el trabajo en el período que se había comprometido.

Antes, las programaciones impuestas por el departamento de gestión raramente eran satisfactorias. Aunque muchos de ellos no eran realistas, el problema principal fue la falta de compromiso personal por parte de los ingenieros. Jugando un papel activo en hacer un compromiso mutuo con la dirección, este ingeniero fue capaz de establecer una programación razonable. En el proceso, él también se comprometió personalmente.

Tabla 8.2 Compromisos semanales fijos de la Estudiante Y.

Nombre Estudiante Y **Fecha** 30/9/96

Presupuesto Semana #1 Compromisos fijos de tiempo

Tarea	Inform.	Física	Matem.	Inglés	Trabajo				Total
Fecha									
D									
L	9:00–9:50	10:00–10:50			4:00–6:00				220
M			9:00–9:50	10:00–10:50	4:00–6:00				220
M	9:00–9:50	10:00–10:50			4:00–6:00				220
J			9:00–9:50	10:00–10:50	4:00–6:00				220
V	9:00–9:50	10:00–10:50			4:00–6:00				220
S									
Totales	150	150	100	100	600				1100

8.5

LA GESTIÓN DE COMPROMISOS NO CONSEGUIDOS

Después de llegar a ser competente haciendo planes, probablemente no fallarás en las fechas de finalización muy a menudo. No puedes, sin embargo, garantizar que nunca vas a fallar en una fecha. Puesto que muchas de las cosas que los ingenieros del software hacen son originales y creativas, hay un riesgo considerable. Si tuvieres en cuenta todos los riesgos posibles, tus estimaciones serían altas de forma poco razonable, de aquí que una programación ocasional o fallos de coste sean inevitables.

Cuando tengas que faltar a un compromiso, notifícalo inmediatamente a la otra parte y poder trabajar juntos para resolver los problemas resultantes. Podrías, por ejemplo, acordar ampliar el plazo de tiempo o reducir la extensión del trabajo. Con el software, una estrategia común es entregar una versión mínima que funcione en la fecha próxima a la programación original y entonces, seguir con una o más mejoras del producto. Definiendo adecuadamente el orden y las fechas de las siguientes funcionalidades, puedes minimizar el trastorno al cliente.

La forma de manejar los errores ocasionales de fecha de entrega o compromisos no cumplidos es importante. Los problemas de los compromisos son siempre desagradables. Probablemente te sentirás mal cuando hayas planificado mal, y la persona con la que te has comprometido se podría sentir engañada o timada. Podría haber consecuencias financieras y la búsqueda de algún culpable. Las cosas desagradables empeoran con el tiempo. Retrasar la notificación con la esperanza de que las cosas mejorarán, normalmente hace que empeoren. Es mejor enfrentarse a lo desagradable tan pronto como entiendas el problema.

Una advertencia importante: no abandones sin intentar seriamente cumplir el compromiso. ¿Lo has discutido con un experto independiente para ver si hay una forma mejor? ¿Podrías añadir recursos para acelerar el trabajo? ¿Hay formas más inteligentes de hacer el diseño? Examina atentamente las alternativas y entonces, si no hay otra forma, enfrente al problema sin dilación.

Los compromisos incumplidos generalmente conducen a molestias e insatisfacciones. Esto es porque, a menudo, la gente retrasa enfrentarse a dichos problemas hasta el Último momento posible; como **los** avestruces, están aplazando lo desagradable con la esperanza de que los problemas desaparezcan de algún modo. Desafortunadamente, meter la cabeza bajo el ala, siempre aumenta la desorganización, reduce las opciones de los clientes y maximiza **los** resultados desagradables.

8.6

LA IMPORTANCIA DE GESTIONAR COMPROMISOS

La principal razón para gestionar tus compromisos, es para no dejarlos pasar u olvidarlos. El trabajo de los ingenieros tiene muchos compromisos. Participan en revisiones, escriben informes, asisten a reuniones, hacen correcciones de programas y actualizan módulos de programas. Pueden tener que: documentar diseños, responder a llamadas de **los** clientes, reunirse con **los** vendedores o participar en comisiones. No es inusual, en el trabajo de los ingenieros, hacer juegos malabares con una docena de compromisos simultáneamente. Es importante aprender a gestionar los compromisos para no romperlos u olvidarlos.

Otra razón para gestionar tus compromisos, es la de ayudarte cuando el trabajo que necesitas hacer, excede el tiempo disponible. Si planificas tu trabajo, no deberías excederte del tiempo con frecuencia, pero puede ser un problema ocasional aun cuando hagas **los** compromisos de forma responsable. En esta situación, identifica rápidamente aquellos compromisos que corren el riesgo de romperse y notifícalo inmediatamente a las otras partes.

8.7

LAS CONSECUENCIAS DE NO GESTIONAR LOS COMPROMISOS

Hasta que no aprendas a gestionar tus compromisos, a menudo te enfrentarás a algunas de las siguientes situaciones desagradables:

El trabajo requerido excede del tiempo disponible. Frecuentemente tendrás más que hacer de lo que puedes realizar. Si no guardas una lista de tus compromisos, puedes obtener nuevos compromisos cuando no deberías hacerlo. Puedes, por ejemplo, comprometerte a una cuestión social cuando debías de hacer unos ejercicios en casa para el próximo día. Por la tarde, recuerdas lo de los ejercicios y entonces, tienes que quedarte toda la noche para hacerlo. Peor aún, puede que ni te acuerdes.

Fallar al enfrentarte a los compromisos. A menudo, los trabajos de desarrollo de software son más complejos de lo que se espera. Cuando no tienes una forma ordenada de establecer compromisos, es probable que asumas que el trabajo es más sencillo de lo que realmente es. Estarás desbordado desde el momento que comiences a realizar el trabajo.

Prioridades mal colocadas. Cuando se está desbordado las personas, a menudo, hacen las prioridades basándose en lo que debe hacerse primero, en vez de lo que es más importante. Cuando se tiene que hacer más de lo que realmente puedes hacer, es natural trabajar sobre la siguiente cosa que debas hacer. Desafortunadamente, resolver la amenaza inmediata, es frecuentemente una estrategia errónea. Cuando estés realmente desbordado, necesitas reestructurar todos tus compromisos, para ajustar lo que puedes hacer. Aplazar o dejar algunas de las tareas inmediatas, puede ser adecuado para hacer los trabajos más importantes que vienen después.

Pobre calidad del trabajo. Trabajando presionados, los ingenieros del software, a menudo, sienten la presión para hacer recortes. Es cuando los descuidos y errores tontos son más probables, y cuando la atención a la calidad es más necesaria. Cuando el tiempo se reduce, los ingenieros deberían tener un cuidado especial para evitar errores. Desafortunadamente, la experiencia demuestra que son en estas circunstancias cuando los ingenieros y directores, probablemente, dedican menos tiempo a hacer revisiones, inspecciones o pruebas completas.

Pérdida de confianza. Si frecuentemente faltas a tus compromisos, la gente lo notará. Sabrán que cuando te comprometes a algo, a menudo, no cumples tu palabra. Tal reputación es difícil de reparar y afectará a tus notas, tu prestigio en el trabajo, tu sueldo e incluso a tu seguridad en el puesto de trabajo.

Pérdida de respeto a tus opiniones. Cuando las personas no confían en lo que tú dices, no es probable que te pidan opinión y es más probable

que te insistan en que tú trabajes con programaciones que no sean razonables.

El activo más importante que puede tener un ingeniero del software es su reputación para cumplir los compromisos. Para que las personas confíen en tu palabra, necesitas exponer tu plan de trabajo y entonces hacer lo que dices. Un propósito importante de este libro es proporcionarte herramientas que te ayuden a hacer compromisos reales que puedas cumplir de forma consecuente.

8.8

LA FORMA DE GESTIONAR COMPROMISOS

Para gestionar adecuadamente los compromisos, comienza haciendo una lista de los compromisos que tienes. Observa la fecha de cada compromiso y la cantidad de tiempo que necesitarás.. La Tabla 8.3 muestra la lista de compromisos de la Estudiante Y.

Tabla 8.3 Lista de compromisos de la Estudiante Y.

Nombre	Estudiante Y	Fecha	30/19/96		
Profesor	Sr. Z	Clase	IP		
<hr/>					
Fecha comprometida	Compromiso	¿Con quién?	Horas	Fecha de	Consigo
Semanal					
LMV	Asistir a clase	Profesor	1,5		Aprobar
LMV	Entregar trabajo inform.	Profesor	6,0		Aprobar
Martes y Jueves	Leer libro	Profesor	4,0		Aprobar
L=>V	Trabajo tiempo parcial, 4:00–6:00 PM	Admisión	10,0	11/9	Paga
Otros					
28/11	Ejercicio trimestral	Profesor	24	11/9	Aprobar

Para gestionar los compromisos en ingeniería del software, es importante recordar varios hechos de la vida del negocio del software.

- Si te estás retrasando, tu planificación continuará retrasándose a no ser que hagas algo diferente.
- Intentarlo más duramente no ayudará. Recuerda, que ya estuviste intentándolo con bastante esfuerzo.

- Si no sabes exactamente dónde estás en el proyecto y cuánto trabajo te queda, sin lugar a dudas estás en un problema.
- Cuando debas tener buena suerte para cumplir un compromiso, no la tendrás.
- Cuando tus estimaciones son erróneas, casi siempre son muy bajas.
- Casi todos los cambios implican más trabajo.

Es importante trabajar enérgicamente para cumplir los compromisos, pero si no puedes terminar el trabajo con unas pocas horas de esfuerzo extra, enfréntate al problema ahora y trátalo con responsabilidad.

RESUMEN

Este capítulo ha definido el compromiso. Ha explicado por qué es importante cumplir los compromisos y cómo gestionarlos. Dos o más partes se comprometen a alguna tarea, cuando están de acuerdo sobre lo que es esa tarea, quién la hará, cuándo se hará y qué se pagará a cambio. Los compromisos que se hacen de forma adecuada, es razonable que se cumplan. Están respaldados por un plan, y tanto el plan como el compromiso están documentados.

Deberías gestionar los compromisos para evitar desbordamientos. Si estás desbordado, te equivocarás en establecer algunos compromisos y ganarás la reputación de ser una persona que no es de fiar. No cumpliendo los compromisos puedes dañar tu carrera, afectará a tus calificaciones y a tu habilidad para conseguir y mantener un trabajo.

Los pasos para gestionar los compromisos son: (1) hacer una lista de tus compromisos actuales, (2) incluir que es lo que hay que hacer y cuando, e (3) incluir una estimación de cuanto trabajo te supondrá cada compromiso.

Recuerda los siguientes hechos en la vida del ingeniero del software: continuarás retrasándote a menos que hagas algo diferente. Intentarlo más duramente no ayudará. Necesitas conocer con precisión dónde estás y qué trabajo te queda por hacer. No puedes contar con la buena suerte para salir de un apuro. Cuando tus estimaciones estén equivocadas, serán casi siempre muy bajas. Casi todos los cambios implican más trabajo.

EJERCICIO 8

Haz una lista de tus compromisos. Incluye un breve resumen sobre en qué consiste el compromiso, quién está comprometido, cuándo se debe cum-

plir, cuánto tiempo esperas dedicarle para cumplirlo, y qué esperas obtener a cambio. Utiliza un formato como el que se muestra en la Tabla 8.3. Entrega una copia de esta lista junto con tu trabajo de casa. Observa que la lista de compromisos es un ejercicio opcional que no necesitas completar a no ser que te lo pida tu profesor.

Entrega las copias de las páginas del cuaderno de registro de tiempos, del cuaderno de trabajos y del resumen semanal de actividades que no hayas entregado todavía.

CAPITULO 9

La gestión de las programaciones

Este capítulo muestra cómo desarrollar y utilizar las programaciones¹ para controlar el progreso del proyecto. Aprenderás a utilizar los puntos de control para controlar el progreso real frente a lo programado; en el ejercicio del capítulo, identificarás y describirás varios puntos de control para controlar el progreso sobre un proyecto de programación.

9.1 LA NECESIDAD DE LAS PROGRAMACIONES

Haces programaciones con objeto de cumplir tus compromisos. Una programación es necesaria cuando tienes varios compromisos de trabajo al mismo tiempo. Con pequeños proyectos o tareas, puedes acabar una tarea antes de comenzar la siguiente. Esta estrategia es factible para unas pocas tareas, cuando hay tiempo suficiente para completarlas todas, y cuando las tareas son cortas o lo bastante sencillas como para acabarlas de un tirón. Cuando las tareas son largas o complejas, o aumenta la carga

¹ No se debe confundir el sentido del término programación como elaboración de un programa de actividades con su sentido como actividad de desarrollo de programas software. Del contexto queda claro que nos referimos a la elaboración de un programa de actividades (N. del T.).

de trabajo, o cuando existen restricciones sobre las programaciones, se necesita una gestión del tiempo más sofisticada. Debes alternar las tareas, haciendo una durante algún tiempo y pasándote entonces a la siguiente.

Supongamos, por ejemplo, que tienes que entregar un programa en tres semanas y un trabajo trimestral que completar en 4 semanas. Supón que prevés que vas a necesitar unas cinco horas para escribir el programa, pero el trabajo debe hacerse en un sistema particular disponible solamente una hora al día. Supón también, que el trabajo trimestral requiere alguna búsqueda bibliográfica y que debes entregar un esbozo en una semana. Esperas que el trabajo trimestral te suponga unas 10 horas de trabajo total. Puesto que planificas dedicar 5 horas a la semana a ambos ejercicios, esperas terminarlos en unas tres semanas.

Siguiendo la estrategia de terminar una tarea antes de comenzar la siguiente, comienzas el trabajo trimestral primero, puesto que el resumen lo debes entregar en una semana. Como esperabas, haces la búsqueda, realizas el esbozo y acabas el resumen en una semana. Entonces, comienzas a programar. Después de trabajar un poco, sin embargo, encuentras que el programa te supondrá alrededor de 10 horas de trabajo en vez de las 5 estimadas. Desafortunadamente, ni con un intenso esfuerzo, puedes tener el ordenador el tiempo suficiente para acabar el programa en el tiempo previsto.

Supón, sin embargo, que has hecho algún trabajo inicial de *ambos* ejercicios durante la primera semana. Entonces verías que el programa era más largo de lo que esperabas y tendrías tres semanas para encontrar el tiempo necesario extra de ordenador.

Conforme trabajas en proyectos cada vez más grandes, será cada vez más importante cuidar la programación de tu tiempo. Un ingeniero del software típico puede tener varios módulos que comprobar para el desarrollo de un proyecto, mientras está trabajando simultáneamente en un informe técnico sobre la documentación de un producto. Pueden surgir cuestiones sobre un programa desarrollado previamente, necesitando reuniones técnicas y de revisión del proyecto. Los ingenieros de algún modo pueden encajar muchas de dichas tareas en sus programaciones diarias sin olvidar o menospreciar cualquiera de ellas. También pueden, a menudo, coordinar su trabajo con muchas otras personas. Es necesario hacer juegos malabares para no tirar ninguna bola, por ello el ingeniero necesita tener una programación personal.

9.2

EL DIAGRAMA DE GANTT

Una programación es una lista ordenada por tiempos de eventos planificados, generalmente presenta un formato como el que se muestra en la

Figura 9.1. Dicha figura se denomina **Diagrama de Gantt** y tiene los siguientes elementos clave:

- En la parte superior del diagrama están las fechas. Dependiendo del grado de detalle deseado, la línea de tiempos, se puede dividir en días, semanas, meses o años. También puede estar en horas si lo deseas.
- La columna que está más a la izquierda contiene un número de identificación (ID) para cada tarea.
- En la segunda columna de la izquierda están los nombres de las tareas a realizar. Están normalmente enumeradas en el orden en el cual esperas hacerlas.
- En el cuerpo del diagrama, las barras muestran las fechas previstas de comienzo y fin para cada tarea.
- En la parte inferior izquierda, está el nombre del proyecto, el autor de la programación y la fecha en que se hizo la programación.
- Varios puntos de control se muestran mediante pequeños óvalos (ID's 2, 5, 7, 9 y 14). Los puntos de control se discutirán posteriormente en este capítulo.

Como puedes observar, el diagrama de Gantt es una forma útil de presentar el flujo general de las tareas de un proyecto. Dichos diagramas son particularmente útiles para coordinar múltiples actividades. Como se explica más adelante, el diagrama de Gantt también proporciona una forma útil de seguir el progreso del proyecto frente al plan y la programación.

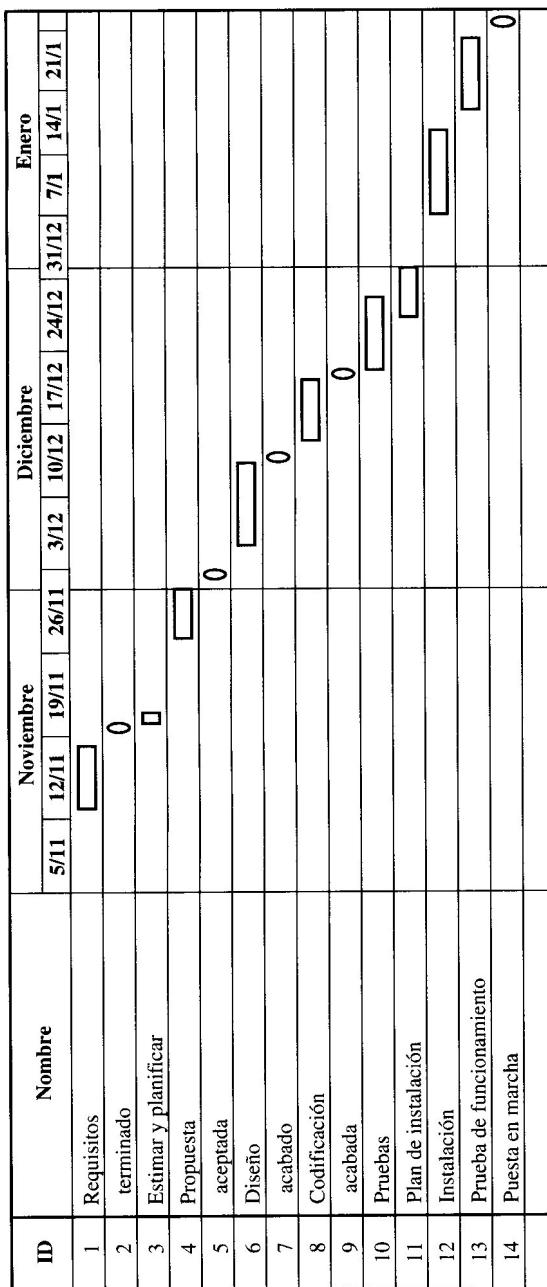
9.3

CÓMO HACER UNA PROGRAMACIÓN DE UN PROYECTO

Para un proyecto de cualquier tamaño, el primer paso para hacer una programación es analizar el trabajo con bastante detalle para identificar las distintas tareas que lo componen. A continuación, estimar el tamaño para cada una de estas pequeñas tareas y determinar la cantidad de trabajo que probablemente necesitarán. Finalmente, registra cada tarea en el diagrama de Gantt con una barra, para mostrar cuándo comienzan y acaban.

Cuando elaboras programaciones para trabajar que implican a varias personas, necesitas seguir algunos pasos adicionales:

1. Asegúrate de que cada individuo conoce las tareas que tiene que hacer.
2. Obtén un compromiso de fechas para cada una de estas tareas.
3. Identifica las interdependencias entre las tareas. ¿Qué información necesita cada persona antes, para ser capaz de realizar el trabajo y de quién le tiene que llegar esa información?



Proyecto: Proyecto ABC
 Autor: Ingeniero X
 Fecha: 12/11/95

Figura 9.1 Ejemplo de diagrama de Gantt.

4. Documenta cada una de estas interdependencias.
5. Revisa la programación propuesta y las interdependencias con todas las personas implicadas, asegúrate de que no hay conflictos, desacuerdos o malentendidos.
6. Revisa la programación para asegurarte de que cubre todas las tareas necesarias para completar todo el trabajo.

Aunque, necesitarás más pasos para elaborar programaciones en grandes proyectos de software, estos principios básicos te ayudarán a hacer planes para tu trabajo personal o para proyectos con equipos pequeños que impliquen a tus compañeros de clase o colegas.

9.4 PUNTOS DE CONTROL

En la planificación de cualquier proyecto, por pequeño que sea, es importante dividir el trabajo en varias partes que puedan ser estimadas y planificadas. Cada una de estas partes se puede tratar como un elemento de la programación. Es decir, cuando se completa cada parte, se ha realizado un determinado grado de progreso. Estos puntos de la programación que son medibles se llaman puntos de control o hitos. Los hitos son una parte importante de la planificación y gestión de proyectos.

Un **hito** es un punto que, objetivamente, se puede identificar en un proyecto. Un ejemplo, sería la terminación de alguna actividad específica del proyecto o una acción importante. Cuando un plan incluye varios hitos, cada uno con una fecha de terminación planificada, puedes ver fácilmente si estás dentro de lo programado o estás retrasándote.

Para ser útiles, los hitos deben ser claros y no ambiguos. Es decir, un punto de control debe ser una acción específica que se hace o no se hace. Algunos ejemplos de puntos de control concretos son:

- Has terminado y entregado un trabajo trimestral.
- Has elaborado y documentado el plan para escribir un programa, utilizando un formato normalizado de planificación.
- Has revisado el plan de desarrollo con tu profesor y has hecho las modificaciones sugeridas.
- Has completado y documentado un diseño de un programa, utilizando un formato de diseño especificado.
- Has implementado, compilado y corregido un programa hasta que se compila sin errores.

Hay muchos ejemplos de puntos de control concretos que son adecuados para los propósitos de la planificación. El requisito clave, es que la terminación de cada punto de control se pueda verificar objetivamente.

Las informaciones generales que no conducen a un criterio verificable no pueden utilizarse como puntos de control.

Puntos de control no adecuados son:

- Has hecho un plan para escribir un programa.
- Has diseñado un programa.
- Se ha completado el 90% de la codificación.

Aunque algunas de estas sentencias podrían refinarse posteriormente, para conducir a un criterio de un punto de control directo y no ambiguo, tal y como están escritas son muy generales. Para la sentencia, “has hecho un plan para escribir un programa”, jcómo podrías saber si el plan contiene la información necesaria? Aunque pudieras encontrar un plan, el punto de control por sí mismo no te dice lo que quieres saber. Si no utilizas un marco de trabajo que defina la planificación, el plan puede estar o no completo. La segunda sentencia, “has diseñado un programa”, jsabes lo que constituye el diseño completo? Casi cualquier cosa se puede considerar como diseño, desde un sencillo diagrama de flujo hasta una descripción detallada. Si se añade a dicha sentencia que el diseño ha de ser documentado en un formato especificado, esto sí podría ser un punto de control adecuado.

Fred Brooks describió la sentencia “La codificación se ha completado hasta un 90%”, como ejemplo de un informe vago y engañoso sobre el estado de un proyecto típico de muchos proyectos software [Brooks]. Si alguien te dice que un proyecto va bien, porque la codificación está casi hecha, es un signo seguro de que hay problemas. Si la persona te dice que ha completado, revisado, compilado y corregido el código de 7 de los 9 nuevos módulos y ha comenzado a trabajar en el octavo, puedes estar tranquilo de que esa persona sabe de lo que está hablando.

SUGERENCIAS PARA EL ESTABLECIMIENTO DE PUNTOS DE CONTROL

En la realización de un plan, establecer puntos de control para cualquier proyecto implica unas pocas horas de trabajo. Intenta identificar varios puntos de control durante una semana. Por ejemplo, si un trabajo te supone unas dos semanas, establece al menos dos puntos de control y preferiblemente cuatro o cinco. Escoge un número de puntos de control coherente con la cantidad de trabajo requerido. Más de un punto de control para una tarea que requiera de tres a cuatro horas sería excesivo.

He encontrado más Útil establecer un punto de control por cada cinco horas de trabajo. Más puntos de control te llevaría mucho más tiempo para controlar actividades insignificantes. Para tareas de menos de cinco horas de duración, gestiona la tarea como una unidad completa. Con es-

tas condiciones, la gestión de proyectos reduce el tiempo de gestión. Conforme vayas dedicando las horas necesarias, irás aproximando el trabajo a lo programado. Deberías, permitir siempre un tiempo reducido de seguridad para cada tarea, para el caso de que te lleve más tiempo de lo esperado.

Para las tareas que tengan una duración de varias semanas, establece al menos un punto de control por cada semana, si solamente esperas dedicarle aproximadamente una media hora semanal a la tarea. Este punto es necesario para recordarte que hagas el trabajo que está planificado. Sin dichos recordatorios, es fácil olvidarse de estas pequeñas tareas. Si realmente no puedes definir un punto de control para cada uno de estos períodos de media hora, lo mejor que puedes hacer es gestionar la cantidad de tiempo que tú decidas. Hasta que realmente completes un determinado punto de control, solo puedes estimar el progreso de forma general.

Los grandes proyectos, a menudo, suponen varias semanas o meses, e implican a varios ingenieros del software. Puesto que las tareas en el proyecto son interdependientes y el proyecto tiene dependencias con otros proyectos, no puedes acabar tus tareas hasta que los otros ingenieros completen las suyas. Es importante para cada ingeniero en un proyecto, tener varios puntos de control intermedios, para que todos puedan saber el estado de todos los trabajos. Si algunos ingenieros tienen problemas, otros pueden ayudarles o adaptar su trabajo para prevenir retrasos en el proyecto global.

9.5

EL SEGUIMIENTO DE LOS PLANES DE LOS PROYECTOS

El seguimiento de un proyecto es una parte importante de la gestión de proyectos y una habilidad clave para los ingenieros del software. Mientras el seguimiento de tu trabajo puede ayudarte con pequeñas tareas, la razón principal de hablarte de este tema ahora, es para mostrarte las técnicas de planificación y control que necesitarás posteriormente. Las habilidades que aprendas con este libro, las aplicarás directamente a los trabajos de gran tamaño que harás como ingeniero del software.

El seguimiento de un plan del proyecto te permite determinar si el proyecto va, adelantado o retrasado según lo programado. Define puntos de control y un plan detallado que pueda descubrir con precisión qué partes de un proyecto están teniendo problemas y dónde se necesita ayuda para ajustarse a lo programado. Estos asuntos se vuelven críticos conforme aumenta el tamaño del proyecto. También se volverán más significativos conforme las repercusiones de faltar a un compromiso se vuelven más serias. Informar sobre el estado real es esencial cuando los proyectos se hacen para los clientes, que son los que pagan.

Otra razón para controlar los planes, es para ser capaz de actuar a tiempo frente al problema. Con un efectivo seguimiento del proyecto, reconocerás los problemas muy pronto, verás cómo corregirlos y, a menudo, serás capaz de recuperar el proyecto. En efecto, un buen sistema de seguimiento puede ayudarte a anticiparte a los problemas antes de que se vuelvan lo bastante serios como para amenazar el éxito del proyecto.

UN EJEMPLO

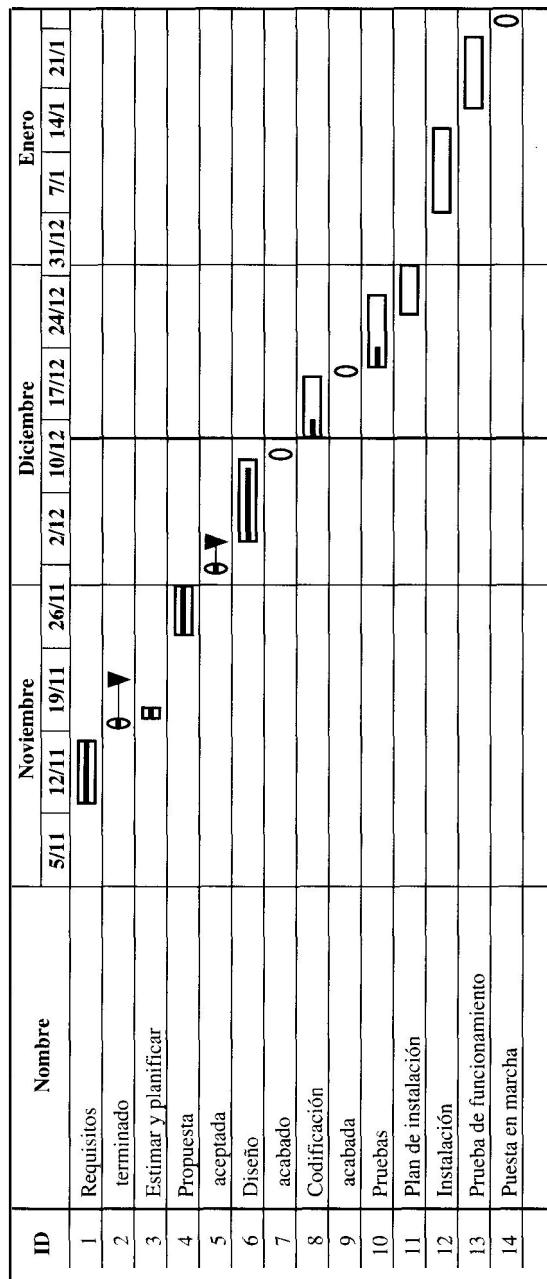
Un diagrama de Gantt se puede utilizar tanto para representar la programación del proyecto como para informar del progreso frente a lo programado. En la Figura 9.2, por ejemplo, el Ingeniero X informa sobre el estado del proyecto el 13 de diciembre. Observa lo siguiente:

- La fecha actual se indica por medio de una línea vertical doble en el día 13/12.
- Las actividades acabadas se indican por medio de una línea horizontal sobre el bloque de la tarea. Por ejemplo, los requisitos (ID 1) se han acabado.
- Las actividades parcialmente acabadas se indican con una línea horizontal parcial sobre el bloque de la tarea. Por ejemplo, el diseño (ID 6) debería haber sido acabado el 12/12, pero muestra solamente una terminación del 80%. Hasta acabarlo, naturalmente, este 80% puede ser solamente una estimación.
- Las actividades del proyecto que llevan ventaja sobre lo programado se indican con una línea de progreso que se extiende más allá de la fecha de evaluación. Por ejemplo, el Ingeniero X comenzó la codificación y las pruebas antes (ID's 8 y 10).
- Los puntos de control acabados se marcan con una flecha en la fecha real de su realización (ID's 2 y 5).

Con esta actualización puedes ver que el Ingeniero X va un poco retrasado en el diseño con respecto a lo programado, pero va un poco adelantado con la codificación y pruebas. El hito “Diseño acabado” aún no se ha alcanzado, por lo que el proyecto se puede considerar que lleva retraso.

ALGUNAS SUGERENCIAS SOBRE LA REVISIÓN DE LA PROGRAMACIÓN

El principal riesgo con la revisión de la programación, es que las personas pueden engañarse a ellas mismas. Pueden tener una falsa visión optimista, fijándose en puntos de control vagos o cambiando frecuentemente la programación. Algunos pasos que te ayudarán a evitar la auto-decepción son:



Proyecto: Proyecto ABC
 Autor: Ingeniero X
 Fecha: 12/1/95
 Actualizado: 13/12/95

Figura 9.2 Estado del diagrama de Gantt.

- Asegúrate de que los puntos de seguimiento están definidos con claridad y están por escrito.
- No cambies la programación hasta que hagas un nuevo plan.
- Cuando informes de la situación real frente al plan, no cambies el plan.
- Si deseas mostrar una nueva estimación de fechas de finalización, deja la programación original en el mismo lugar y anota las nuevas fechas con líneas de puntos.
- Guarda copias de la programación original y de todas las actualizaciones.

La clave consiste en recordar que estás comparando la situación real frente a la programación original que no has cambiado. Dicha programación era el plan que estás midiendo frente al trabajo realizado. Si alteras el plan original, no tendrás nada frente a lo que medir. Esto es por lo que, cuando estás controlando el estado del proyecto, es una buena idea comprobar la fecha inicial de la programación. Si dicha fecha está muy próxima a la fecha de revisión, entonces la actualización no proporciona una medida útil del progreso del proyecto. Pide una programación original.

9.6

EL SEGUIMIENTO DEL VALOR CONSEGUIDO

Un problema que surge con el seguimiento del proyecto es que, a menudo, es difícil saber donde estás. Supón, por ejemplo, que estás siguiendo mi progreso en la escritura del manuscrito del libro. El plan original para los diez primeros capítulos se muestra en la Tabla 9.1. Como puedes ver, el trabajo comenzó con una planificación y un esbozo del esfuerzo, a lo que esperaba dedicarle unos 681 minutos. Siguiendo la planificación, escribí el prólogo y cada capítulo. La tarea implicaba escribir un borrador del capítulo, corregir el borrador y reescribirlo. El tiempo esperado para cada capítulo se muestra en la Tabla 9.1. Aquí, el tiempo que esperaba dedicar a cada tarea se muestra en la columna Plan-Minutos y la fecha que esperaba para la finalización se muestra bajo la columna Semana-Plan. La semana 1, comencé el lunes 17 de abril de 1995. Este plan estaba basado en los datos que había reunido mientras estaba escribiendo mis dos libros anteriores.

Después de completar este plan, me entrevisté con el profesorado de informática de la Universidad Aeronáutica de Embry-Riddle, que querían utilizar el manuscrito del libro para impartir cursos en el mes de septiembre de ese año. Eso significaba que necesitaba completar el manuscrito de los 10 primeros capítulos para el 10 de agosto. Como se ve en la Tabla 9.1, el plan no mostraba la terminación del capítulo 10 hasta la semana 21, que comenzaba el día 4 de septiembre, o sea, tres semanas más tarde.

Tabla 9.1 Plan de desarrollo del manuscrito.

Capítulo		Plan Minutos	Valor Planificado		Semana		Valor Conseguido
			Unidad	Acumu.	Plan	Real	
Plan		681	3,2431	3,243	1	1	3,244
Prólogo	Borrador	207	1,3671	4,610	2	1	1,367
	Editar	000	3,0101	0,420	2	2	3,010
	Reescribir	760	3,6191	12,0391	3		
			1,3671	13,4061	3	2	1,367
	Editar	000	3,0101	17,216	4	2	3,010
	Reescribir	760	3,6191	20,835	4		
Capítulo 2	Borrador	207	1,3671	22,202	4	2	1,367
	Editar	000					
	Reescribir	760	3,6191	29,631	6		
Capítulo 3	Borrador	207	1,3671	30,998	7	3	1,367
	Editar	000	3,0101	34,000	7	3	3,010
			3,6191	30,427	0		
Capítulo 4	Borrador	207	1,3671	39,794	10	3	1,367
	Editar	000	3,0101	43,6041	10	4	3,010
	Reescribir	760	3,6191	47,223	10		
Capítulo 5	Borrador	287	1,3671	48,590	11	4	1,367
	Editar	000					
	Reescribir	760	3,6191	56,019	12		
Capítulo 6	Borrador	207	1,3671	57,386	12	4	1,367
	Editar	000	3,0101	61,196	14		
Capítulo 7	Borrador	207	1,3671	66,102	15	4	1,367
	Editar	000	3,0101	69,9921	16		
	Reescribir	760	3,6191	73,611	17		
			1,3671	74,970	17		
	Editar	800	3,0101	70,700	18		
	Reescribir	760	3,6191	02,407	10		
Capítulo 9	Borrador	207	1,3671	03,774	19		
	Editar	000	3,0101	07,505	19		
	Reescribir	760	3,6191	91,203	20		
Capítulo 10	Borrador	207	1,3671	92,570	20		
	Editar	000					
			3,6191	100,0	21		
Total		20.990	100,0				

Antes que rehacer el plan, decidí revisar mi trabajo de unas semanas, para ver si podía hacer una nueva programación más reducida. También decidí, que en vez de completar cada capítulo, uno tras otro, yo escribiría y corregiría los borradores, y que algún editor asociado los revisaría antes de la redacción final. Esta aproximación tendría la ventaja de informarme sobre las críticas del trabajo, y probablemente redundaría en un mejor producto final. La desventaja era que no podría completar la revisión de cualquier capítulo hasta muy tarde. Entonces, jcómo podría decir si estaba actuando sobre lo programado?

La respuesta a este problema es algo llamado **valor conseguido**. Básicamente, examiné el plan y asigné un valor a cada tarea basándome en la cantidad de tiempo que esperaba dedicarle, de acuerdo con lo planificado. Hice esto calculando para cada tarea el porcentaje correspondiente del total de los 20.998 minutos planificados para esta parte del proyecto. Así, la tarea planificada de 681 minutos tendría un valor ganado de $100 * 681 / 20.998 = 3,243\%$. Para conseguir la programación acelerada, tenía que alcanzar el valor del 100% el día 10 de agosto, o la semana 17, en vez de la semana 21.

Después de las cuatro primeras semanas de trabajo, pude ver que completaría los 10 primeros capítulos a tiempo (véanse la Tabla 9.2 y la Figura 9.3). Aquí, calculé el valor estimado, anotando lo que había ganado de valor durante cuatro semanas (37,037), o 9,259 cada semana. A

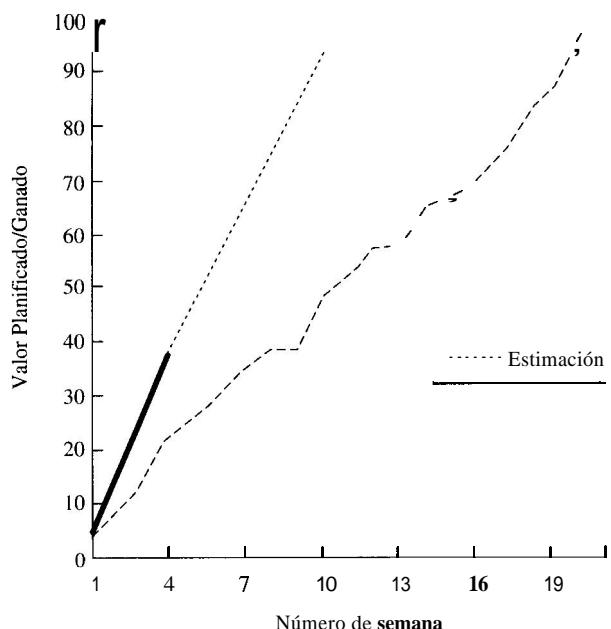


Figura 9.3 Manuscrito del libro. Valores reales y estimados.

Tabla 9.2 Programación del libro.

Semana	Valor Planificado	Valor Ganado	Valor Estimado
1	3,243	4,610	
2	8,420	14,963	
3	13,406	25,317	
4	22,202	37,037	
5	26,012		46,296
6	29,631		55,556
7	34,808		64,815
8	38,427		74,074
9	38,427		83,333
10	47,223		92,592
11	52,400		101,852
12	57,306		
13	57,386		
14	64,815		
15	66,182		
16	69,992		
17	74,978		
18	82,407		
19	87,585		
20	96,380		
21	100,0		

esta velocidad, podía esperar conseguir los valores que se muestran en la columna de Valores Estimados de la Tabla 9.2. Aquí, aparece que alcanzaría el 100% la semana 11, o sea mucho antes de la semana 17 que era la semana obligada.

El caso es que, utilizando el valor ganado, podía hacer el trabajo en un orden diferente al planificado originalmente y todavía controlar el progreso frente al plan. Aunque podía haber rehecho el plan, habría sido mucho trabajo y habría retrasado el proyecto. Puesto que había añadido la tarea de revisión, la terminación llevaba un poco más que lo planificado y realmente acabé el manuscrito en la semana 15, a tiempo para el próximo semestre.

Aunque no se requiere utilizar más adelante en este libro, el seguimiento del valor ganado lo encontrarás una técnica muy útil para poste-

riores gestiones de programaciones en grandes proyectos. El valor conseguido se discute más a fondo en el capítulo 6 del libro general del PSP [Humphrey].

RESUMEN

Este capítulo cubre la programación y los puntos de control, explicando porqué las programaciones son necesarias y cómo utilizar los diagramas de Gantt para hacer programaciones. Los puntos de control son una parte importante de la planificación y gestión de un proyecto: un punto de control identifica la terminación de un determinado evento del proyecto, se debe de especificar y establecer el criterio de terminación. Tendrás al menos un punto de control cada 5 o 10 horas de trabajo, y al menos uno o dos puntos de control para cada proyecto semanal. Los puntos de control te ayudan a ver dónde se encuentra el proyecto y a tomar acciones correctivas cuando el proyecto se está retrasando. El valor ganado te ayuda a controlar con precisión el estado del proyecto y estimar exactamente cuándo acaba.

EJERCICIO 9

Define tres o más puntos de control para el trabajo de codificar un programa de tamaño reducido. Entrega un breve informe describiendo cada punto de control y como estos puntos te podrían indicar cuánto se ha avanzado. Observa que el ejercicio del punto de control es opcional y que no lo necesitas hacer a no ser que te lo pida tu profesor.

Entrega las copias de las páginas del Cuaderno de Registro de Tiempos, del Cuaderno de Trabajos y del Resumen Semanal de Actividades que no hayas entregado todavía.

REFERENCIAS

[Brooks] Brooks, Frederick P. *The Mythical Man-Month, Essays on Software Engineering, Anniversary Edition*. Reading, MA: Addison-Wesley, 1995.

[Humphrey] Humphrey, W.S. *A Discipline for Software Engineering*. Reading, MA: Addison-Wesley, 1995.

CAPITULO 10

El plan del proyecto

Este capítulo extiende la discusión de la planificación del producto, para incluir estimaciones de tiempo. Se introduce la tabla del Resumen del Plan del Proyecto para registrar, lo estimado y lo real, del tamaño del programa y el tiempo de desarrollo. Como ejercicio, elaborarás un plan del proyecto para el siguiente programa que desarrolles.

10.1

LA NECESIDAD DE LOS PLANES DE LOS PROYECTOS

El plan del proyecto define el trabajo y cómo se hará. Proporciona una definición de cada tarea principal, una estimación del tiempo y de los recursos necesarios y un marco de trabajo para gestionar la revisión y el control. El plan del proyecto, es también, un poderoso vehículo de aprendizaje. Cuando está adecuadamente documentado, es un punto de referencia para comparar con el rendimiento real. Esta comparación permite a los planificadores ver sus errores de estimación y mejorar su exactitud en la planificación [Humphrey 89].

El plan del proyecto es una parte esencial del mismo. En un proyecto grande, lo más importante es tener un plan. La razón de que hagas planes de proyectos en este curso, es para que aprendas a hacerlos. Conforme adquieras más experiencia, **tu** capacidad para elaborar planes exactos mejorará y serás capaz de hacerlos rápidamente cuando los necesites. Por ahora, el plan del proyecto que harás, solamente incluirá una estimación

del tamaño del producto y el tiempo que estimas en hacer el trabajo. Después de acabar el trabajo, también escribirás los datos reales de tamaño y tiempo.

Una vez que has hecho una estimación de tiempo, escribe la fecha de finalización del trabajo, y así, puedes comparar lo estimado con la fecha real de realización del mismo. Analizar dichas comparaciones, te ayudará a hacer mejores estimaciones para las tareas futuras. Si por ejemplo, tus estimaciones normalmente son muy bajas, puedes ajustarlas al alza, o a la inversa, si las estimaciones son muy altas, puedes ajustarlas a la baja. Ten cuidado al hacer estos ajustes, aunque sean fáciles de corregir.

10.2 EL RESUMEN DEL PLAN DEL PROYECTO

La tabla que utilizarás para registrar los datos planificados y reales del proyecto se muestra en la Tabla 10.1. En capítulos posteriores se añadirán otros elementos a esta tabla. Para evitar el tener que introducir nuevas tablas en cada capítulo, utilizaremos ahora la tabla Resumen del Plan del Proyecto mostrada en la Tabla 10.2, en la cual las entradas que posteriormente se introducirán aparecen sombreadas. Ignora ahora lo sombreado.

Para completar la tabla, rellena las partes de **Plan**, antes de comenzar el proyecto de codificación. Cuando acabes, completa los datos en la

Tabla 10.1 Resumen del plan del proyecto.

Estudiante _____	Fecha _____	
Programa _____	Programa # _____	
Profesor _____	Lenguaje _____	
Resumen Minutos/LOC LOC/Hora	Plan _____	Real _____
Tamaño Programa(LOC): Total Nuevo & Cambiado	Plan _____	Real _____
Tamaño Máximo Tamaño Mínimo	_____	_____
Tiempo por Fase(min.) Total	Plan _____	Real _____
Tiempo Máximo Tiempo Mínimo	_____	_____

Tabla 10.2 Resumen del plan del proyecto del PSP.

Estudiante _____	Fecha _____		
Programa _____	Programa # _____		
Profesor _____	Lenguaje _____		
Resumen	Plan	Real	Hasta la Fecha
Minutos/LOC	_____	_____	_____
LOC/Hora	_____	_____	_____
Defectos/KLOC	_____	_____	_____
Rendimiento	_____	_____	_____
VIF	_____	_____	_____
Tamaño Programa (LOC):			
Total Nuevo & Cambiado	_____	_____	_____
Tamaño Máximo	_____	_____	_____
Tamaño Mínimo	_____	_____	_____
Tiempo por Fase (min.)	Plan	Real	Hasta la Fecha % Hasta la Fecha
<i>Planeación</i>	_____	_____	_____
<i>Diseño</i>	_____	_____	_____
<i>Codificación</i>	_____	_____	_____
<i>Revisión del código</i>	_____	_____	_____
<i>Compilación</i>	_____	_____	_____
<i>Pruebas</i>	_____	_____	_____
<i>Postmorten</i>	_____	_____	_____
Total	_____	_____	_____
Tiempo Máximo	_____	_____	_____
Tiempo Mínimo	_____	_____	_____
Defectos introducidos	Plan	Real	Hasta la Fecha % Hasta la Fecha Def./Hora
<i>Planeación</i>	_____	_____	_____
<i>Diseño</i>	_____	_____	_____
<i>Codificación</i>	_____	_____	_____
<i>Revisión del código</i>	_____	_____	_____
<i>Compilación</i>	_____	_____	_____
<i>Pruebas</i>	_____	_____	_____
Total	_____	_____	_____
Defectos eliminados	Plan	Actual	Hasta la Fecha % Hasta la Fecha Def./Hora
<i>Planeación</i>	_____	_____	_____
<i>Diseño</i>	_____	_____	_____
<i>Codificación</i>	_____	_____	_____
<i>Revisión del código</i>	_____	_____	_____
<i>Compilación</i>	_____	_____	_____
<i>Pruebas</i>	_____	_____	_____
Total	_____	_____	_____

parte de **Real**. En los ejercicios del capítulo, te pedirán que presentes una tabla rellena del Resumen del Plan del Proyecto por cada programa que termines a partir de ahora. Guarda copias de estas tablas para que tengas datos de tu trabajo, y puedas utilizarlos en futuros ejercicios. El siguiente apartado explica como llenar la tabla del Resumen del Plan del Proyecto.

10.3 EL RESUMEN

La sección Resumen contiene los datos de la velocidad utilizados para hacer el plan. También, proporciona un lugar para registrar la velocidad real después de acabar el trabajo. La primera entrada en la sección Resumen es Minutos/LOC (minutos por líneas de código). Bajo la columna Plan, escribe la velocidad planificada en Minutos/LOC. Como se muestra en el ejemplo de la Tabla 10.3, el Estudiante X utiliza los datos históricos de

Tabla 10.3 Ejemplo del resumen del plan del proyecto.

Estudiante	Estudiante X	Fecha	7/10/96
Programa		Programa#	8
Profesor	Sr. Z	Lenguaje	Ada
Resumen	Plan	Real	Hasta la Fecha
Minutos/LOC	7,82	7,21	
LOC/Hora	7,67	8,32	
Defectos/KLOC			
Rendimiento			
V/F			
<i>Tamaño Programa (LOC):</i>			
Total Nuevo & Cambiado	26	19	
Tamaño Máximo	36		
Tamaño Mínimo	18		
<i>Tiempopor Fase (min.)</i>	Plan	Real	Hasta la Fecha % Hasta la Fecha
<i>Planificación</i>			
<i>Diseño</i>			
<i>Codificación</i>			
<i>Revisión del código</i>			
<i>Compilación</i>			
<i>Pruebas</i>			
<i>Postmorten</i>			
Total	203	137	
Tiempo Máximo	282		
Tiempo Mínimo	141		
<i>Defectos introducidos</i>	Plan	Actual	Hasta la Fecha % Hasta la Fecha Def./Hora
<i>Planificación</i>			
<i>Diseño</i>			
<i>Codificación</i>			
<i>Revisión del código</i>			
<i>Compilación</i>			
<i>Pruebas</i>			
Total			
<i>Defectos eliminados</i>	Plan	Actual	Hasta la Fecha % Hasta la Fecha Def./Hora
<i>Planificación</i>			
<i>Diseño</i>			
<i>Codificación</i>			
<i>Revisión del código</i>			
<i>Compilación</i>			
<i>Pruebas</i>			
Total			

sus programas anteriores o del Cuaderno de Trabajos para tomar la velocidad de 7,82 Minutos/LOC para este programa. Naturalmente, si no tienes datos históricos, tendrás que inventártelos.

A no ser que tengas una buena razón para hacerlo de otra manera, utiliza tus datos históricos de velocidad media de los últimos programas registrados en el Cuaderno de Trabajos. Puedes tener necesidad de utilizar un valor diferente, si un nuevo programa es difícil y necesita más tiempo de la media. O puedes necesitar utilizar un valor diferente para el caso inverso, si un nuevo programa es parecido a otro que has escrito recientemente pero que probablemente necesite menos esfuerzo. Comprobando los valores de tiempo medio, máximo y mínimo para los trabajos de codificación más recientes, puedes seleccionar un valor adecuado para un nuevo programa.

La siguiente entrada en la sección Resumen es LOC/Hora (líneas de código por hora). De nuevo, escribe el valor planificado antes de comenzar el trabajo y el valor real después de terminarlo. El valor de LOC/Hora se calcula a partir de los Minutos/LOC, dividiendo 60 por el valor de Minutos/LOC. En el ejemplo de la Tabla 10.3, el Estudiante X planificó una velocidad de 7,82 Minutos/LOC, por lo que el valor de LOC/Hora planificado es $60/7,82 = 7,67$. El valor de la velocidad de LOC/Hora lo utilizan normalmente los ingenieros para analizar la productividad del desarrollo.

10.4 EL TAMAÑO DEL PROGRAMA

La sección Tamaño del Programa (LOC) de la tabla Resumen del Plan del Proyecto, contiene los datos estimados y reales de los tamaños de los programas y de los rangos de dichos tamaños. Utilizando los métodos descritos en el Capítulo 6, estima el tamaño total de un programa que planificas desarrollar y escríbelo en la columna Plan de la fila Total Nuevo & Cambiado. Para completar el Resumen del Plan del Proyecto para el programa 8, el Estudiante X hizo una estimación del tamaño, tal y como se muestra en la tabla 10.4.

La razón de que la casilla del tamaño esté etiquetada como Total Nuevo & Cambiado, es porque deberías registrar solamente los números de LOC que escribas realmente. Después de escribir unos pocos programas, comenzarás a utilizar la librería de rutinas para hacer funciones estándar, o desarrollar un nuevo programa añadiendo o modificando uno hecho previamente. Para ahorrar tiempo de desarrollo, puedes copiar partes de programas hechos anteriormente. Puedes, por ejemplo, decidir reutilizar una función en un programa que ya tengas escrita. Antes que escribir todo de nuevo, sencillamente copia ese código y reutilízalo en el nuevo programa.

Tabla 10.4 Estimación del tamaño del programa del Estudiante X.

Estudiante Profesor	Estudiante X Sr. Z			Fecha Clase	7/10/96 IP		
Programa	LOC	Funciones anteriores	Funciones estimadas	Mín.	Media	Máx.	
		Sentencia case sencilla					
Bucles		Sentencia case media	Sentencia casegrande	12	17	24	
		Bucle while sencillo	Bucle <i>pequeño</i>	1 3	1 4	5	
Calc.		Repetir-hasta medio					
		Cálculo grande					
Texto							
	5	7	Cadena pequeña de texto	Cadena sencilla de texto	3	5	7
	7	24	Cadena media de texto				
Estimado							
				18	26	36	

Comentarios: Este programa tiene una sentencia *case* bastante grande para seleccionar entre un número de condiciones de texto, su tamaño podría ser un poco mayor que la del programa 4.

Se necesita un bucle sencillo para iterar las condiciones de la sentencia *case* para cada cadena que entra, también se necesita un sencillo analizador de cadenas de texto para formatear el texto y obtener el resultado final. Supón que los tamaños máximo para las funciones del bucle y del texto son similares a los de programas anteriores, y el de la sentencia *case* es algo mayor.

Asume también que el tamaño mínimo es la mitad del tamaño máximo.

Ninguna de las líneas de código desarrolladas anteriormente son contabilizadas cuando estimas el tamaño del nuevo programa. La razón es que tú quieras controlar las LOC que realmente escribes. Ya que estas líneas son las que consumen la mayor parte del tiempo y con ellas se calcula la velocidad de productividad en Minutos/LOC. Esta fila, en el Resumen del Plan del Proyecto, se denomina Total Nuevo & Cambiado, porque este es el único tipo de código a contabilizar, tanto las LOC nuevas escritas como cualquiera de las anteriores que se hayan cambiado. Entonces, para estimar cuánto tiempo necesitarás para desarrollar un programa nuevo, multiplica los Minutos/LOC planificados por las LOC Nuevas & Cambiadas planificadas.

Por ejemplo, si copias 25 LOC de un programa anterior y desarrollas 30 LOC de código nuevo, solamente tendrás 30 LOC Nuevas & Cambiadas.

Por otra parte, si copias 25 LOC de un programa anterior y modificas 7 de esas LOC, serían 37 las LOC Nuevas y Cambiadas: 7 LOC del código cambiado y 30 LOC de código nuevo.

La razón de contabilizar solamente las LOC Nuevas & Cambiadas de esta forma es que, la cantidad de código que utilizas de las librerías y de programas anteriormente desarrollados variará mucho. También, el tiempo, expresado en minutos/LOC, para incluir código, es generalmente inferior al que se requiere normalmente para desarrollar código nuevo o modificar el existente. Así, si ignoras las LOC reutilizadas y copiadas, generalmente ignorarás una parte relativamente pequeña del trabajo total y tus estimaciones probablemente serán bastante exactas. Aunque la forma de contabilizar el código modificado y reutilizado puede ser muy compleja, por ahora incluirás todas las LOC nuevas y cambiadas en las estimaciones de tamaño, pero no el código reutilizado sin modificar. Ignora cualquier LOC tomada de varios sistemas de librería o de programas previamente desarrollados. Pero si el tiempo dedicado en la reutilización de programas previamente desarrollados parece mucho para ignorarlo, consulta un tratamiento más detallado de este tema en los Capítulos 4 y 5 del libro general del PSP [Humphrey 95].

TAMAÑO MÁXIMO Y MÍNIMO

Los valores de tamaño máximo y mínimo en la sección Tamaño del Programa (LOC) se obtienen a partir de un tamaño estimado, que se hace de la forma que se describió en el Capítulo 6. En el ejemplo de la Tabla 10.4, el Estudiante X utilizó la suma de los tamaños de los programas 1 y 5 como los tamaños máximos para dos de las funciones y seleccionó un valor algo mayor del programa 4 para la otra función. Para calcular el mínimo, asumió que las nuevas funciones combinadas deberían ser la mitad del tamaño total máximo. Entonces, selecciona el tamaño más probable como el valor medio entre esos números. Así, tomó 36 como el tamaño máximo y 18 LOC como el mínimo, y 26 LOC como el tamaño más probable.

Los valores de tamaño máximo y mínimo son útiles para considerar el rango de tiempo probable para estimar un desarrollo. Por ejemplo, si estás particularmente interesado en completar un programa nuevo en una fecha acordada, considera el tamaño máximo como una indicación de cómo de grande podría ser el programa. Si has estimado que el tamaño probable para el nuevo programa era de 26 LOC y el tamaño máximo de 36 LOC, puedes permitirte alguna libertad en la codificación, en caso de que el programa se aproxime hacia el valor máximo.

El valor del tamaño mínimo se obtiene como se describe en el Capítulo 6. La razón principal para calcular este número es animarte a pensar sobre el rango de tamaño de un programa planificado. Generalmente,

basa los planes y los compromisos en los valores de tamaño máximo y mínimo.

Observa también, que los valores máximo y mínimo están basados únicamente en los rangos de tamaño estimado. No se deducen estadísticamente y no se pueden tomar para proporcionarte límites estadísticos. Hasta que no tengas una cantidad sustancial de datos, solamente utiliza las estimaciones de tamaño máximo y mínimo como guía de planificación general. Cuando tengas más datos, considera utilizar los cálculos de predicción de intervalos dados en el Capítulo 5 y en el Apéndice A del libro general del PSP [Humphrey 95].

10.5 EL TIEMPO EN FASES

La siguiente sección de la Tabla 10.3 del Resumen del Plan del Proyecto se denomina Tiempo en Fase, ya que posteriormente se utiliza para los datos de las fases del proceso de desarrollo del software. Hasta el próximo capítulo, sin embargo, utiliza solamente el tiempo total de desarrollo del programa.

Para calcular el tiempo total planificado para el desarrollo de un nuevo programa, estima el tamaño del programa en LOC y entonces multiplícalo por los Minutos/LOC planificados de la parte superior de la tabla. Esto te da una estimación de los minutos totales para desarrollar el programa. Estos valores también se muestran en el ejemplo de la Tabla 10.3. El tiempo total planificado se obtiene multiplicando los Minutos/LOC planificados por las LOC Nuevas & Cambiadas planificadas: $7,82 \times 26 = 203,32$, o 203 minutos.

También calcula los tiempos máximo y mínimo y escríbelos en la columna de Plan. Obtén estos valores multiplicando los tamaños máximo y mínimo por los Minutos/LOC para calcular los tiempos máximo y mínimo respectivamente. Para el mínimo tenemos $7,82 \times 18 = 140,76$ y para el máximo, $7,82 \times 36 = 281,52$.

Observa que tú también podrías obtener los valores máximo/máximo y mínimo/mínimo multiplicando las LOC máximas y mínimas por las velocidades máxima y mínima del Cuaderno de Trabajos. Puesto que estos valores, generalmente, proporcionan límites amplios para utilizarlos de guía, nosotros no los utilizaremos en el Resumen del Plan del Proyecto. Pero pueden dar alguna idea de los mejores y peores tiempos de desarrollo que puedes esperar.

Completa y escribe estos datos del Plan en el Resumen del Plan del Proyecto antes de desarrollar el programa y entonces completa los datos Reales cuando acabes el programa. Conforme hagas el trabajo, revisa el tiempo y cuando acabes escribe el total en la fila de Total bajo la colum-

na Real. También, cuenta las LOC Nuevas & Cambiadas en el programa terminado y calcula los Minutos/LOC y las LOC/hora reales. Para revisar la forma de llenar totalmente el Resumen del Plan del Proyecto consulta las instrucciones en la Tabla 10.5.

Tabla 10.5 Instrucciones del resumen del plan del proyecto.

Propósito	Esta tabla trata los datos estimados y reales de los proyectos de una forma cómoda y fácilmente recuperable.
Cabecera	<p>Introduce los siguientes datos:</p> <ul style="list-style-type: none"> • Tu nombre y fecha de hoy. • Nombre y número de programa. • Nombre del profesor. • El lenguaje que utilizarás para escribir el programa.
Resumen Minutos/LOC	<p>Antes de iniciar el desarrollo:</p> <ul style="list-style-type: none"> • Escribe los Minutos/LOC planificados para este proyecto. <p>Después del desarrollo:</p> <ul style="list-style-type: none"> • Divide el tiempo total de desarrollo por el tamaño real del programa para obtener los Minutos/LOC reales. • Por ejemplo, si el proyecto se hizo en 137 minutos e hiciste 19 LOC, los Minutos/LOC serían $137/19 = 7,21$.
LOC/Hora	<p>Antes de iniciar el desarrollo:</p> <ul style="list-style-type: none"> • Calcula las LOC por hora planificada para este programa dividiendo 60 por los Minutos/LOC de la casilla de Plan. <p>Después del desarrollo:</p> <ul style="list-style-type: none"> • Para LOC/Hora Real divide 60 por los Minutos/LOC Reales. • Para los 7,21 Minutos/LOC Reales, tenemos $60/7,21 = 8,32$ LOC/Hora Reales
Tamaño Programa (LOC)	<p>Antes de iniciar el desarrollo:</p> <ul style="list-style-type: none"> • Escribe bajo la columna plan el valor estimado de Total Nuevas & Cambiadas, Máximo y Mínimo. <p>Después del desarrollo:</p> <ul style="list-style-type: none"> • Cuenta y escribe las LOC Nuevas & Cambiadas Reales.
Tiempo por fase	<p>Para el tiempo total de desarrollo, multiplica el valor de Total Nueva & Cambiada LOC por los Minutos/LOC.</p> <p>Por ejemplo, con 7,82 Minutos/LOC y Total Nueva & Cambiada LOC de 26 LOC, $Total = 26 * 7,82 = 203$ minutos.</p> <p>Para el tiempo Máximo, multiplica el tamaño Máximo por los Minutos/LOC.</p> <p>Por ejemplo, con 7,82 Minutos/LOC y un tamaño Máximo de 36 LOC, $Tiempo Máximo = 36 * 7,82 = 282$ minutos.</p> <p>Para el tiempo Mínimo, multiplica el tamaño Mínimo por los Minutos/LOC.</p> <p>Por ejemplo, con 7,82 Minutos/LOC y un tamaño Mínimo de 18 LOC, $Tiempo Mínimo = 18 * 7,82 = 141$ minutos.</p>
Real	<p>Una vez acabado el trabajo, escribe el tiempo de desarrollo real en minutos.</p> <p>Obtén estos datos del Cuaderno de Tiempos.</p>

10.6**CÓMO ESTIMAR CON EXACTITUD**

Tus estimaciones iniciales de tamaño y tiempo probablemente no serán muy precisas al principio. Esto es normal. El primer objetivo, es aprender a hacer estimaciones imparciales. Es decir, por cada 10 estimaciones, será deseable que 5 estén por arriba y otras 5 por abajo. Aunque preferirías que todas fuesen correctas, dicha exactitud no es probable. Las fluctuaciones en las estimaciones se reducirán gradualmente, pero deberías ser capaz rápidamente de hacer estimaciones imparciales, es decir, hacer más o menos las mismas sub y sobreestimaciones. Con experiencia, puedes gradualmente reducir la desviación media del error de la estimación.

Documenta tus estimaciones, estúdialas y aprende de ellas. Esto te ayudará a hacer mejores estimaciones. Algunos ingenieros estiman mejor que otros. Conforme sean más precisas tus estimaciones, aprenderás cómo es de grande la tolerancia para hacer sub y sobreestimaciones. Esto te ayudará a estimar el riesgo de hacer compromisos que no puedas cumplir.

RESUMEN

Este capítulo muestra como hacer planes de proyectos. Describe el proceso de planificación del producto, define los datos que necesitas y las tablas, y muestra un ejemplo completo de un plan de proyecto. Documentar los planes del proyecto te permite comparar el tiempo y el tamaño estimados con el de desarrollo real y aprenderás a hacer mejores planes.

Para hacer buenos planes, necesitas una estimación de la cantidad de trabajo a realizar comparándola con trabajos anteriores, datos históricos de tareas similares que hayas hecho, y una tabla para registrar el plan. Para obtener los datos para hacer planes, mide los tamaños de los productos acabados, estima el tamaño del siguiente trabajo, estima la velocidad de desarrollo para este trabajo, calcula el tiempo que te supondrá el nuevo trabajo, y determina sus rangos probables de tamaño y tiempo de desarrollo.

EJERCICIO 10

Haz un plan para codificar el próximo programa. Basa este plan en tus datos históricos y utiliza la Tabla 10.2 de planificación. En el futuro, haz y entrega un plan para cada ejercicio de codificación. En este plan, registra la estimación inicial antes de comenzar el desarrollo y escribe los tiempos y tamaños reales cuando hayas acabado el trabajo. También, entrega

una copia de la estimación del tamaño del programa, para ello utiliza la Tabla 10.4.

Entrega una copia de los Cuadernos de Tiempos, del Cuaderno de Trabajos y del Resumen Semanal de Actividades que no hayas entregado todavía. A partir de ahora y hasta el final del semestre, entrega estas tablas llenas para cada programa que desarrolles durante este curso.

REFERENCIAS

[Humphrey 89] Humphrey, W.S. *Managing the Software Process*. Reading, MA: Addison-Wesley, 1989.

[Humphrey 95] Humphrey, W.S. *A Discipline for Software Engineering*. Reading, MA: Addison-Wesley, 1995.

CAPITULO 11

El proceso de desarrollo del software

Seguir un proceso sistemático, te ayudará a planificar y gestionar tu trabajo más efectivamente. En este capítulo se explica cómo utilizar un proceso para desarrollar software. Como ejercicio, harás un plan y codificarás un programa, utilizando el proceso descrito en este capítulo.

11.1 **POR QUÉ UTILIZAMOS LOS PROCESOS**

Cuando he colaborado con organizaciones de software para mejorar su rendimiento, uno de los principales problemas que he encontrado, es la determinación del rendimiento real de las mismas. Un grupo, por ejemplo, no podía decirme cuántos proyectos se habían entregado tarde o habían superado el presupuesto. Sin dichos datos, no había forma de decirles si ellos iban mejorando o empeorando. Para centrar este problema, comenzaron a reunir datos utilizando los métodos que aprenderás en este capítulo.

Un proceso es un conjunto definido de pasos para hacer un trabajo. Cada paso o fase de un trabajo tiene especificado unos criterios de entrada que deben ser satisfechos antes de comenzar la fase. De forma similar, cada fase tiene unos criterios de salida que deben satisfacerse antes de ter-

minar la fase. Los pasos del proceso definen las tareas y cómo se hacen. El diseño y gestión de procesos son importantes en ingeniería del software, porque la calidad del proceso del ingeniero, determina en gran parte, la calidad y productividad de su trabajo. El objetivo del proceso personal definido en este libro es ayudarte a ser más efectivo como ingeniero del software.

El proceso utilizado en este curso se denomina Proceso Software Personal (PSP). El PSP es un marco de trabajo que ayuda a los ingenieros del software a medir y mejorar su forma de trabajar. Estos dos objetivos son los que te ayudan a desarrollar programas y a mostrarte, como utilizando los procesos, puedes mejorar tu forma de trabajar. Posteriormente, cuando desarrolles programas más grandes y complejos, puedes extender el PSP para ayudarte con ese trabajo. Este libro no describe cómo definir y extender los procesos. Todo esto se cubre en el texto general del PSP [Humphrey 95].

11.2 ALGUNAS DEFINICIONES

Para discutir los procesos, utilizamos los términos definidos en el Capítulo 5 y algunos pocos más:

- Un **producto** es algo que produces para un colaborador, un empresario o un cliente.
- Un **proyecto** normalmente produce un producto.
- Una **tarea** se define como un elemento de trabajo.
- Un **proceso** define la forma de hacer proyectos.
- Los procesos tienen varias **fases** o pasos, como la planificación, el desarrollo y las pruebas.
- Una fase de un proceso puede estar compuesta de múltiples tareas o actividades, como pruebas de integración, pruebas del producto y pruebas del sistema.
- Obsérvese que un proceso puede tener solamente una o muchas fases, y una fase puede tener solamente una o muchas tareas o actividades.
- Los planes describen la forma en que un proyecto concreto va a ser hecho: cómo, cuándo y qué coste tendrá. También puedes planificar tareas individuales.
- Un **trabajo** es algo que haces, tanto un proyecto como una tarea.

Cuando un proceso está totalmente descrito, se denomina *proceso definido*. Los procesos definidos están compuestos normalmente de guiones, ta-

blas, plantillas y estándares. Un guión del proceso es una conjunto de pasos escritos, que los usuarios o agentes del proceso siguen cuando utilizan el proceso. Distintas tablas, tales como registros y resúmenes, se utilizan para registrar y almacenar los datos del proyecto. En el resto de capítulos del libro, utilizarás el PSP. Los elementos del PSP se muestran en la Figura 11.1. Observa que, aunque esta figura muestre tanto los tiempos como los cuadernos de los defectos, no comenzarás a reunir datos de defectos hasta el Capítulo 12.

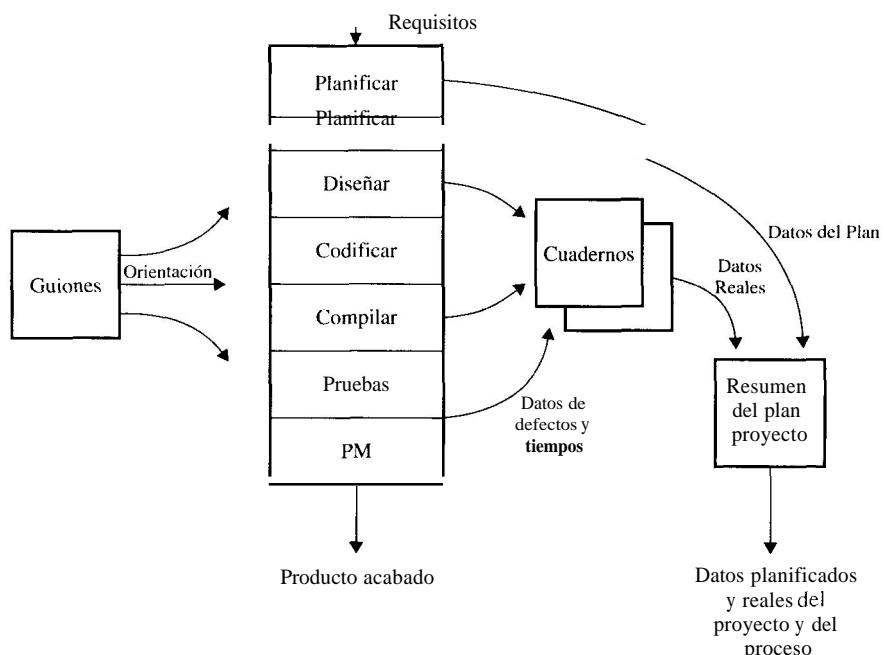


Figura 11.1 Flujo del proceso del PSP

11.3 EL GUIÓN DEL PROCESO

El guión inicial del PSP se presenta en la Tabla 11.1. Esta es la primera de las distintas versiones del guón del PSP. En los capítulos siguientes, este guón se extiende para incluir pasos adicionales. Las fases del proceso del PSP se describen a continuación y están resumidas en la Tabla 11.1.

Planificación. Primero obtén los requisitos del proyecto y completa las partes no sombreadas del Plan en la tabla Resumen del Plan del Proyecto. Finalmente, escribe el tiempo dedicado a hacer esta planificación en el Cuaderno de Registro de Tiempos.

Tabla 11.1 Guión del proceso del PSP.

	Propósito	Guia en el desarrollo de pequeños programas.
	Criterios de entrada	La descripción del problema. Tabla Resumen del Plan del Proyecto del PSP. Datos de tamaños y tiempos reales de programas anteriores. Cuaderno de Registro de Tiempos.
1	Planificación	Obtén una descripción de las funciones del programa. Estima las LOC Máx., Mín. y total requeridas. Determina los Minutos/LOC. Calcula los tiempos de desarrollo Máx., Mín., y total. Escribe los datos del plan en la tabla Resumen del Plan del Proyecto. Anota el tiempo de planificación en el Cuaderno de Registro de Tiempos.
2	Diseño	Diseña el programa. Anota el diseño en el formato especificado. Anota el tiempo de diseño en el Cuaderno de Registro de Tiempos.
3	Codificación	Implementa el diseño. Utiliza un formato estándar para introducir el código. Anota el tiempo de codificación en el Cuaderno de Registro de Tiempos.
4	Compilación	Compila el programa. Corrige todos los errores encontrados. Anota el tiempo de compilación en el Cuaderno de Registro de Tiempos.
5	Pruebas	Prueba el programa. Corrige todos los errores encontrados. Anota el tiempo de pruebas en el Cuaderno de Registro de Tiempos.
6	Postmortem	Completa la tabla de Resumen del Plan del Proyecto con los datos de tiempo y tamaño reales. Anota el tiempo postmortem en el Cuaderno de Registro de Tiempos.
	Criterios de salida	Programa probado a fondo. Diseño adecuadamente documentado. Listado completo del programa. Resumen del Plan del Proyecto completado. Cuaderno de Registro de Tiempos completado.

Diseño. Diseña el programa. Aunque el diseño necesario no se haga, piensa toda la lógica del programa antes de comenzar a escribir el código. Guarda el diseño en un diagrama de flujo, pseudocódigo o en cualquier formato que especifique tu profesor. Al final de la fase de diseño, anota el tiempo de diseño en el Cuaderno de Registro de Tiempos. El proceso de

diseño y las distintas clases de representación del mismo se discutirán posteriormente en el Capítulo 17.

Codificación. Implementa el diseño codificándolo en el lenguaje de programación seleccionado. Utiliza un formato de codificación consistente y sigue los estándares de codificación que especifique tu profesor. Al final de la fase de codificación, anota el tiempo de codificación en el Cuaderno de Registro de Tiempos.

Compilación. Compila el programa y corrige todos los defectos que encuentres. Continúa compilando y corrigiendo los defectos hasta que compiles el programa sin ningún mensaje de error. Todo el tiempo dedicado a esta fase se contabiliza como tiempo de compilación, aunque corrijas el código o cambies el diseño. Al final de la fase de compilación, anota el tiempo de compilación en el Cuaderno de Registro de Tiempos.

Pruebas. Haz bastantes pruebas para asegurarte que los programas cumplen todos los requisitos y superan un conjunto adecuado de pruebas sin error. Todo el tiempo que dediques a esta fase se contabiliza como tiempo de prueba, incluyendo la corrección del código, el cambio de diseño y la re-compilación. Al final de esta fase, anota el tiempo de pruebas en el Cuaderno de Registro de Tiempos.

Postmortem. Completa los datos reales en la tabla Resumen del Plan del Proyecto, tal y como se describe en el resto de este capítulo. Puesto que has registrado el tiempo postmortem antes de que hayas acabado la fase postmortem, completa el trabajo como puedas y entonces dedica unos pocos minutos para hacer los cálculos finales y escribirlos en el tiempo postmortem estimado. Utiliza este tiempo postmortem estimado para calcular el tiempo de desarrollo total y el resto de cálculos.

11.4

PUNTOS DE CONTROL Y FASES

Los puntos de control fueron introducidos en el Capítulo 9 para ayudar a hacer y controlar las programaciones de los proyectos. Definiendo de forma explícita y clara los puntos de control del proyecto, puedes hacer planes mejores. La causa de por qué estos planes son mejores, se debe a que los puntos de control proporcionan unos puntos de referencia precisos para medir el estado del proyecto mientras estás haciendo el trabajo.

El proceso de desarrollo del software, extiende la idea de punto de control desde unos pocos puntos a todas las fases del proceso. Con un proceso definido, cada fase produce un resultado específico y por lo tanto la conclusión de una fase es un punto de control medible. Utilizando un proceso definido, tendrás muchos puntos de control para ayudarte a la planificación y revisión de tu trabajo.

11.5 ACTUALIZACIÓN DE LA TABLA RESUMEN DEL PLAN DEL PROYECTO

El Resumen del Plan del Proyecto de la Tabla 11.2, es una de las tablas para el proceso del PSP. Sus instrucciones aparecen en la Tabla 11.3. Como hemos visto antes, algunas partes del Resumen del Plan del Proyecto están sombreadas. Estas partes pueden ignorarse ahora porque

Tabla 11.2 Resumen del plan del proyecto del PSP.

Estudiante _____	Fecha _____		
Programa _____	Programa # _____		
Profesor _____	Lenguaje _____		
Resumen	Plan	Real	Hasta la fecha
Minutos/LOC	_____	_____	_____
LOC/Hora	_____	_____	_____
Defectos/KLOC	_____	_____	_____
Rendimiento	_____	_____	_____
VF	_____	_____	_____
Tamaño Programa (LOC):			
Total Nuevo & Cambiado	_____	_____	_____
Tamaño Máximo	_____	_____	_____
Tamaño Mínimo	_____	_____	_____
Tiempo por Fase (min.)	Plan	Real	Hasta la Fecha % Hasta la Fecha
Planificación	_____	_____	_____
Diseño	_____	_____	_____
Codificación	_____	_____	_____
Revisión del código	_____	_____	_____
Compilación	_____	_____	_____
Pruebas	_____	_____	_____
Postmorten	_____	_____	_____
Total	_____	_____	_____
Tiempo Máximo	_____	_____	_____
Tiempo Mínimo	_____	_____	_____
Defectos introducidos	Plan	Real	Hasta la Fecha % Hasta la Fecha Def./Hora
Planificación	_____	_____	_____
Diseño	_____	_____	_____
Codificación	_____	_____	_____
Revisión del código	_____	_____	_____
Compilación	_____	_____	_____
Pruebas	_____	_____	_____
Total	_____	_____	_____
Defectos eliminados	Plan	Real	Hasta la Fecha % Hasta la Fecha Def./Hora
Planificación	_____	_____	_____
Diseño	_____	_____	_____
Codificación	_____	_____	_____
Revisión del código	_____	_____	_____
Compilación	_____	_____	_____
Pruebas	_____	_____	_____
Total	_____	_____	_____

Tabla 11.3 Instrucciones del resumen del plan del proyecto del PSP.

Propósito	Esta tabla trata los datos estimados y reales de los proyectos de una forma cómoda y fácilmente recuperable.
Cabecera	Introduce los siguientes datos: <ul style="list-style-type: none"> • Tu nombre y fecha de hoy. • Nombre y número de programa. • Nombre del profesor. • El lenguaje que utilizarás para escribir el programa.
Minutos/LOC	Antes de iniciar el desarrollo <ul style="list-style-type: none"> • Escribe los Minutos/LOC planificados para este proyecto. <i>Utiliza la velocidad Hasta la Fecha de un programa reciente del Cuaderno de Trabajos del Resumen del Plan del Proyecto.</i> Después del desarrollo <ul style="list-style-type: none"> • Divide el tiempo total de desarrollo por el tamaño real del programa para obtener los Minutos/LOC reales. • Por ejemplo, si el proyecto se realizó en 196 minutos e hiciste 29 LOC, los Minutos/LOC serían $196/29 = 6,76$.
LOC/Hora	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> • Calcula las LOC por hora planificada para este programa dividiendo 60 por los Minutos/LOC de la casilla de Plan. Después del desarrollo <ul style="list-style-type: none"> • Para LOC/Hora Real divide 60 por Minutos/LOC Reales. • Para los 6,76 Minutos/LOC Reales, tenemos $60/6,76 = 8,88$ LOC/Hora Reales
Tamaño Programa (LOC)	Antes de iniciar el desarrollo <ul style="list-style-type: none"> • Escribe bajo la columna plan, el valor estimado de Total Nuevas & Cambiadas, Máximo y Mínimo. Después del desarrollo: <ul style="list-style-type: none"> • Cuenta y escribe las LOC Nuevas & Cambiadas Reales. • <i>Para la columna Hasta la Fecha, añade LOC Reales Nuevas & Cambiadas a las LOC Hasta la Fecha Nuevas & Cambiadas de programas anteriores.</i>
Tiempo por fase Plan	Para el tiempo total de desarrollo, multiplica el valor de las LOC Total Nuevas & Cambiadas por los Minutos/LOC. Para el tiempo Máximo, multiplica el tamaño Máximo por los Minutos/LOC. Para el tiempo Mínimo, multiplica el tamaño Mínimo por los Minutos/LOC. <i>Del Resumen del Plan del Proyecto de un programa reciente, busca los valores de % Hasta la Fecha para cada fase. Utilizando el % Hasta la Fecha de programas anteriores calcula el tiempo planificado para cada fase.</i>
Real	Una vez acabado el trabajo, anota el tiempo real en minutos que has gastado <i>en cada fase del desarrollo</i> . Obtén estos datos del Cuaderno de Tiempos.
Hasta la fecha	<i>Para cada fase, escribe la suma del tiempo real y el tiempo Hasta la Fecha de los programas más recientes.</i>
% hasta la fecha	<i>Para cada fase, escribe 100 multiplicado por el tiempo Hasta la Fecha y lo divides por el total del tiempo Hasta la Fecha.</i>

no se utilizarán hasta capítulos posteriores. Para identificar los cambios de una versión del proceso a la siguiente, los añadidos se muestran en **negrita**. Comparando la Tabla 10.2 y la 11.2, puedes observar que varias secciones que antes estaban sombreadas ahora se utilizan. Por ejemplo, bajo la columna Tiempo por Fase, las filas de Planificación, Diseño, Codificación, Compilación, Pruebas y Postmortem se incluyen ahora en la Tabla. También, se utilizan las columnas Hasta la Fecha y % Hasta la Fecha. Para indicar que estas columnas se añaden de nuevo a la tabla, las cabeceras de la nueva columna y fila aparecen en **negrita**.

La sección de Tiempo por Fase de la nueva tabla del Resumen del Plan Proyecto, tiene una fila por cada fase del proceso. Esta fila tiene los tiempos planificados y reales para cada fase del proceso. Durante la fase de planificación, escribe los datos bajo la columna Plan. Durante la fase Postmortem, escribe los datos bajo la columna Real. Cuando anotes el tiempo en el Cuaderno de Registro de Tiempos, escribe en la sección de comentarios la fase del proceso donde estés. Entonces, durante la fase postmortem, escribe estos tiempos en la columna Real de la sección Tiempo en Fase para cada fase.

Antes de iniciar un proyecto, completa la parte de Plan de la tabla Resumen del Plan Proyecto, tal y como se indica en el Capítulo 10. Ahora, la única diferencia es que tú necesitas estimar el tiempo que vas a dedicar a cada fase. Esto se hace asignando a cada fase un porcentaje del tiempo de desarrollo total, basándote en la utilización del tiempo en proyectos anteriores. La primera vez que utilices esta versión del PSP, no tendrás datos reales para hacer esto, entonces, tendrás que inventártelos. Para proyectos sucesivos, sin embargo, puedes utilizar los datos de proyectos anteriores para estimar el tiempo para cada fase del nuevo proyecto. Esta es la razón de por qué aparecen los valores de % Hasta la Fecha en la tabla Resumen del Plan del Proyecto.

Las columnas de Hasta la Fecha y % Hasta la Fecha de la tabla Resumen del Plan del Proyecto, proporcionan una forma sencilla de calcular la distribución de los porcentajes del tiempo de desarrollo para las fases del proceso. La columna Hasta la Fecha contiene el total de todos los tiempos dedicados en cada fase para todos los programas acabados. La columna de % Hasta la Fecha tiene el porcentaje de los tiempos de la columna Hasta la Fecha. El ejemplo de la Sección 11.7 muestra cómo calcular las entradas de Hasta la Fecha y % Hasta la Fecha.

11.6

UN EJEMPLO DE PLANIFICACIÓN

La Tabla 11.4 muestra como el Estudiante X ha completado parte del plan de la tabla Resumen del Plan del Proyecto para el programa 9. Utilizó los

Tabla 11.4 Resumen del plan del proyecto del PSP.

Estudiante	Estudiante X		Fecha	21/10/96
Programa			Programa#	9
Profesor	Sr. Z		Lenguaje	Ada
Resumen			Hasta la fecha	
Minutos/LOC	7,21			
LOC/Hora	8,32			
Defectos/KLOC				
Rendimiento				
V/F				
Tamaño Programa (LOC):				
Total Nuevo & Cambiado	23			
Tamaño Máximo				
Tamaño Mínimo	15			
Tiempo por Fase (min.)	Plan	Real	Hasta la Fecha	% Hasta la Fecha
Planificación	=			
Diseño				
Codificación	74			
Revisión del código				
Compilación	25			
Pruebas	52			
Postmorten	10			
Total	166			
Tiempo Máximo	224			
Tiempo Mínimo	108			
Defectos introducidos	Plan	Real	Hasta la Fecha	% Hasta la Fecha
Planificación				
Diseño				
Codificación				
Revisión del código				
Compilación				
Pruebas				
Total				
Defectos eliminados	Plan	Real	Hasta la Fecha	% Hasta la Fecha
Planificación				
Diseño				
Codificación				
Revisión del código				
Compilación				
Pruebas				
Total				

datos del Resumen del Plan del Proyecto del programa 8 de la Tabla 11.5. Los datos del plan de la Tabla 11.4 se obtienen de la siguiente forma:

- *Minutos/LOC.* En la planificación del programa 9, busca los Minutos/LOC Reales para el programa anterior, el programa 8 de la Tabla 11.5, y localiza la velocidad de 7,21 Minutos/LOC.

A no ser que tengas una buena razón para hacer otra cosa, utiliza estas velocidades reales anteriores en la planificación de los nuevos proyec-

Tabla 11.5 Resumen del plan del proyecto del PSP.

Estudiante	Estudiante X	Fecha	7/10/96
Programa		Programa#	8
Profesor	Sr. Z	Lenguaje	Ada
<i>Resumen</i>			
Minutos/LOC	7,82	Real	7,21
LOC/Hora	7,67		8,32
<i>Defectos/KLOC</i>			
<i>Rendimiento</i>			
VIF			
<i>Tamaño Programa (LOC):</i>			
Total Nuevo & Cambiado	26		19
Tamaño Máximo	36		
Tamaño Mínimo	18		
<i>Tiempo por Fase (min.)</i>	Plan	Real	Hasta la Fecha
Planificación	10	4	4
Diseño	19	0	0
Codificación	110	61	61
<i>Revisión del código</i>			% Hasta la Fecha
Compilación	12	21	21
Pruebas	29	43	43
Postmorten	15	0	0
Total	203	137	137
Tiempo Máximo	222		
Tiempo Mínimo	141		
<i>Defectos introducidos</i>	Plan	Real	Hasta la Fecha
Planificación			% Hasta la Fecha
Diseño			Def./Hora
Codificación			
Revisión del código			
Compilación			
Pruebas			
Total			
<i>Defectos eliminados</i>	Plan	Real	Hasta la Fecha
Planificación			% Hasta la Fecha
Diseño			Def./Hora
Codificación			
Revisión del código			
Compilación			
Pruebas			
Total			

tos. En el futuro, mejor que utilizar la velocidad real de los programas anteriores, utilizarás las velocidades media de todos los programas desarrollados hasta la fecha. Cuando se incluyan las velocidades Hasta la Fecha en el PSP en el Capítulo 12, podrás utilizar estos datos.

- *LOC/Hora.* El Estudiante X calculó el valor de LOC/Hora, así: $60/7,21=8,32$.
- *Tamaño del programa.* De igual forma que en el Capítulo 6, el Estudiante X estimó las LOC Total Nuevas & Cambiadas del pro-

grama y las LOC Máx. y Mín. En el ejemplo de la Tabla 11,4, estos tamaños son 23, 31 y 15 LOC respectivamente.

- *Tiempo por Fase/Total.* Utilizando el tamaño estimado de 23 LOC y la velocidad de 7,21 minutos/LOC, el tiempo de desarrollo es $7,21 \times 23 = 166$ minutos.
- *Tiempo Máximo.* El tiempo máximo se obtiene multiplicando el valor de los Minutos/LOC por el tamaño máximo, $7,21 \times 31 = 224$ minutos.
- *Tiempo Mínimo.* El tiempo mínimo se obtiene multiplicando el valor de los Minutos/LOC por el tamaño mínimo, $7,21 \times 15 = 108$ minutos.

Con un tiempo total de desarrollo estimado de 166 minutos, utiliza los datos de % Hasta la Fecha de la Tabla 11.5 para estimar los tiempos de las fases para desarrollar el programa 9. El Estudiante X hizo estos cálculos de la siguiente forma:

- *Planificación.* El tiempo de planificación estimado es $2,9 \times 166/100 = 4,81$, o aproximadamente 5 minutos.
- *Diseño.* El tiempo de diseño planificado es $0 \times 166/100 = 0$.
- *Codificación.* El tiempo de codificación planificado es $44,6 \times 166/100 = 74,04$, o 74 minutos.
- *Compilación.* El tiempo de compilación planificado es $15,3 \times 166/100 = 25,40$, o 25 minutos.
- *Pruebas.* El tiempo de pruebas planificado es $31,4 \times 166/100 = 52,12$, o 52 minutos.
- *Postmortem.* El tiempo postmortem planificado es $5,8 \times 166/100 = 9,63$, o 10 minutos.

Recuerda que del primer programa desarrollado con el PSP, no tendrás datos anteriores para utilizarlos como guía. Puesto que el Estudiante X no tenía datos anteriores cuando estaba planificando el programa 8, se los inventó. Como puedes observar, comparando los tiempos planificados y reales de la Tabla 11.5, su distribución real de tiempo era bastante distinta a sus suposiciones. Unos pocos datos pueden provocar una gran diferencia en la planificación. Para el programa 9, el plan del Estudiante X está mucho más próximo a sus datos reales.

11.7

UN EJEMPLO PARA CALCULAR LOS DATOS HASTA LA FECHA

En la fase postmortem del programa 9, el Estudiante X contabilizó 29 LOC nuevas y cambiadas, y anotó este número en la casilla Real de la

Tabla Resumen del Plan del Proyecto de la Tabla 11.6. Después, localizó los tiempos reales para cada fase en el Cuaderno de Registro de Tiempos y los anotó en la tabla Resumen del Plan del Proyecto. Esta vez, ambas estimaciones de tamaño y tiempo eran razonablemente más próximas al resultado real. El Estudiante X le dedicó algo más de lo esperado para el desarrollo del programa, su velocidad LOC/Horas real estaba muy próxima a sus velocidades históricas. También, la distribución de tiempos no se alejó mucho del plan.

Para completar la columna Hasta la Fecha de la Tabla 11.6, el Estudiante X sumó **los** tiempos reales de esta tabla a los tiempos Hasta la Fecha del programa 8 de la Tabla 11.5. También, calculó los porcentajes Hasta la Fecha de la Tabla 11.6, dividiendo los valores de Hasta la Fecha en cada fase por el tiempo Total Hasta la Fecha de 333 minutos y multiplicarlos por 100. Los cálculos para estos valores fueron los siguientes:

- LOC Total Nuevas & Cambiadas LOC Hasta la Fecha Nuevas y Cambiadas = $19 + 29 = 48$.
- Planificación Tiempo de planificación Hasta la Fecha = $4 + 11 = 15$.
% Hasta la Fecha, planificado = $100 * 15 / 333 = 4,5$
- Diseño Tiempo de diseño Hasta la Fecha = $0 + 12 = 12$.
% Hasta la Fecha, diseño = $100 * 12 / 333 = 3,6$.
- Codificación Tiempo de codificación Hasta la Fecha = $61 + 85 = 146$.
% Hasta la Fecha, codificación = $100 * 146 / 333 = 43,9$
- Compilación Tiempo de compilación Hasta la Fecha = $21 + 28 = 49$.
% Hasta la Fecha, compilación = $100 * 49 / 333 = 14,7$.
- Pruebas Tiempo de pruebas Hasta la Fecha = $43 + 49 = 92$.
% Hasta la Fecha, pruebas = $100 * 92 / 333 = 27,6$.
- Postmortem Tiempo postmortem Hasta la Fecha = $8 + 11 = 19$.
% Hasta la Fecha, postmortem = $100 * 19 / 333 = 5,7$.
- Total Tiempo total de desarrollo Hasta la Fecha = $137 + 196 = 333$.

Tabla 11.6 Resumen del plan del proyecto del PSP.

Estudiante	Estudiante X		Fecha	21/10/96
Programa			Programa#	9
Profesor	Sr. Z		Lenguaje	Ada
<i>Resumen</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la fecha</i>	
Minutos/LOC	7,21	6,76		
LOC/Hora	8,32	8,88		
<i>Defectos/KLOC</i>				
<i>Rendimiento</i>				
VIF				
<i>Tamaño Programa (LOC):</i>				
Total Nuevo & Cambiado	23	29	48	
Tamaño Máximo	31			
Tamaño Mínimo	15			
<i>Tiempo por Fase (min.)</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la Fecha</i>	<i>% Hasta la Fecha</i>
Planificación	5	11	15	4,5
Diseño	0	12	12	3,6
Codificación	74	05	146	43,9
<i>Revisión del código</i>				
Compilación	25	20	49	14,7
Pruebas	52	49	92	27,6
Postmorten	10	11	19	5,7
Total	166	196	333	100,0
Tiempo Máximo	224			
Tiempo Mínimo	100			
<i>Defectos introducidos</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la Fecha</i>	<i>% Hasta la Fecha</i>
Planificación				
Diseño				
Codificación				
Revisión del código				
Compilación				
Pruebas				
Total				
<i>Defectos eliminados</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la Fecha</i>	<i>% Hasta la Fecha</i>
Planificación				
Diseño				
Codificación				
Revisión del código				
Compilación				
Pruebas				
Total				

Con los datos del programa 9, el Estudiante X puede estimar los tiempos que va a dedicar en cada fase de su próximo proyecto. Para ser más práctico, sin embargo, deberías utilizar la media de los tiempos de los distintos proyectos. Las columnas Hasta la Fecha y % Hasta la Fecha realizan estos cálculos. En el programa 8 de la Tabla 11.5, la columna Hasta la Fecha contiene los tiempos reales que el Estudiante X ha dedicado al proyecto. Cuando desarrolló el programa 9, sin embargo, añadió los tiempos reales de los programas 8 y 9 para obtener los nuevos valores

Hasta la Fecha. El nuevo valor de % Hasta la Fecha nos da la distribución de tiempos medios de los programas 8 y 9. De forma similar, cuando se complete el Resumen del Plan del Proyecto para el programa 10, tendrás la distribución de tiempos medios para los programas 8, 9 y 10 y así sucesivamente.

De un proyecto a otro, los tiempos de desarrollo total variarán. Sin embargo, la distribución del tiempo entre las fases, será más estable. Esto, naturalmente, depende de la calidad de tus procesos. Por ejemplo, cuando dedicas mucho tiempo a compilar y probar, los tiempos de planificación de las distintas fases serán menos exactos debido a la gran cantidad de tiempo impredecible que se dedica a resolver los defectos. Cuando se obtienen los tiempos medios de los distintos programas, sin embargo, la cantidad media de tiempo dedicado a la compilación y a las pruebas no cambiará mucho. Es decir, no cambiará mientras no cambies el proceso. Cuando al principio utilices el PSP, probablemente dedicarás de 1/3 a 1/2 de tu tiempo a encontrar y corregir los defectos durante la fase de compilación y pruebas. Conforme utilices los métodos del PSP en los siguientes capítulos, reducirás el número de defectos encontrados en la compilación y pruebas, y así reducirás los tiempos de compilación y pruebas. Esto ahorrará tiempo de desarrollo, mejorará la estimación y permitirá planes más exactos y programas mejor hechos.

RESUMEN

Este capítulo define un proceso, describe el proceso básico del PSP y muestra como un proceso definido, puede ayudar a mejorar tus planes. La tabla del Resumen del Plan del Proyecto aumenta para incluir los tiempos de las fases del proyecto y calcular el tiempo Hasta la Fecha y el porcentaje del tiempo de desarrollo dedicado a cada fase. Haciendo planes de proyectos, querrás estimar el tiempo que vas a dedicar a cada fase. Te basarás en experiencias anteriores, utilizando para ello los valores de % Hasta la Fecha de los programas anteriores. Utilizando el PSP, harás un Resumen del Plan del Proyecto para cada proyecto, completa las partes del Plan antes de comenzar a trabajar, escribe los datos Reales cuando acabes y completa los apartados de Hasta la Fecha y % Hasta la Fecha.

EJERCICIO 11

Utilizando la tabla del Resumen del Plan del Proyecto mostrada en la Tabla 11.2, haz un plan para codificar tu próximo programa. Primero, estima el tamaño del programa, tal y como se describe en el Capítulo 6. Para

este primer programa con el PSP, estima la distribución del tiempo en las distintas fases. En adelante, utiliza para hacer estimaciones, las cifras de % Hasta la Fecha de la tabla del Resumen del Plan del Proyecto del programa más recientemente desarrollado. Anota la estimación antes de hacer el trabajo y cuando lo acabes, escribe los tiempos y tamaños reales de las distintas fases.

También, entrega una copia de los Cuadernos de Registro de Tiempos, del Cuaderno de Trabajos y del Resumen Semanal de Actividades que no hayas entregado todavía.

REFERENCIA

[Humphrey 95] Humphrey, W.S. A *Discipline for Software Engineering*. Reading, MA: Addison-Wesley, 1995.

CAPITULO 12

Defectos

En este capítulo se introduce el tema de los defectos del software (véase la Sección 12.4 para saber por qué no le llamamos bugs). Los defectos pueden causar serios problemas a los usuarios de los productos software, y pueden ser caros de encontrar y corregir. Puesto que los defectos son causados por los errores de los desarrolladores, los ingenieros necesitan entender los defectos que introducen para aprender a gestionarlos. El primer paso para la gestión de defectos, es reunir los datos de los defectos que has introducido en tu programa. Con estos datos, puedes mejorar la forma de encontrarlos y corregirlos. Como ejercicio del capítulo, reunirás y resumirás los datos de los defectos que hay en tus programas.

Hasta ahora, en este libro, solamente hemos hablado de métodos para gestionar el coste y las programaciones. Esta, es la mitad de la historia. Comenzando con este capítulo, estudiaremos la necesidad de entregar productos software de calidad. Primero, necesitaremos una definición de calidad.

12.1

¿QUÉ ES LA CALIDAD DEL SOFTWARE?

La calidad del software afecta a los costes de desarrollo, programación de las entregas y la satisfacción del usuario. Puesto que la calidad del software es tan importante, necesitamos discutir primero qué significa la palabra *calidad*. La calidad de un producto software debe ser definida en términos

que tengan significado para los usuarios del producto. Así, un producto que proporciona las prestaciones que son más importantes para los usuarios, es un producto de calidad. Las necesidades de los usuarios, a menudo, se expresan en los documentos de requisitos. Debido a su importancia, el desarrollo, clarificación y refinamiento de los requisitos es un objetivo por si mismo. Por lo tanto, nosotros no vamos a tratar los requisitos en este libro. Es importante recordar, sin embargo, que hasta que no tengas claro los requisitos, no puedes desarrollar un programa de calidad. Aunque puedes comenzar con requisitos poco claros, debes entenderlos antes de poder acabar.

La calidad del software es un tema tan enorme, que en este libro se tratará de forma parcial. El libro, sin embargo, proporciona las habilidades y prácticas que necesitarás para entender los defectos que introduces, y esto te dotará, de un mecanismo eficiente para que encuentres y corrijas muchos de tus defectos. También te proporcionará los datos para ayudar a prevenir estos defectos en el futuro. Finalmente, una vez que puedas gestionar los defectos eficientemente, puedes dedicar más atención a aquellos aspectos de la calidad que afectan a la utilidad y valor de los programas que desarrolles.

12.2

DEFECTOS Y CALIDAD

Recuerda, del Capítulo 1, que el trabajo de un ingeniero del software es entregar productos de calidad con unos costes y programaciones planificadas. Recuerda también, que los productos software deben satisfacer tanto las necesidades funcionales de los usuarios como hacer de una forma segura y consistente el trabajo de los mismos. La realización del trabajo es un aspecto clave. Aunque las funciones del software son muy importantes para los usuarios de los programas, estas funciones no serán útiles a menos que el software funcione. Para que el software funcione, debes eliminar sus defectos. Así, aunque hay muchos aspectos relacionados con la calidad del software, el primer aspecto de la calidad está relacionado necesariamente con sus defectos. Esto no significa que los defectos son el Único aspecto o que son lo más importante, pero debes tratar con muchos de los defectos antes de poder satisfacer cualquiera de los otros objetivos del programa. Después de conseguir que los programas funcionen, si tienes unos pocos defectos, no funcionarán en grandes sistemas, no se utilizarán, y no se tendrá en cuenta sus otras cualidades.

La causa de que los defectos sean tan importantes, es porque las personas cometen muchos errores. En efecto, los programadores experimentados normalmente cometen un error por cada 7 a 10 líneas de código que desarrollan. Aunque generalmente encuentran y corrigen muchos de esos defectos cuando compilan y prueban sus programas, a

menudo, muchos de los defectos permanecen en el producto acabado. Entonces, tu primera prioridad es entender los defectos que introduces y prevenirlos como puedas. Para hacer esto, necesitas dominar el lenguaje de programación que utilices, entender a fondo los sistemas que soportan el desarrollo y haber dominado los tipos de aplicaciones que desarrollarás. Estos y otros pasos más son necesarios para reducir el número de defectos que introduces.

12.3 ¿QUÉ SON LOS DEFECTOS?

El término *defecto* se refiere a algo que está equivocado en un programa, tal como un error sintáctico, una falta tipográfica, un error de puntuación, o una sentencia incorrecta del programa. Los defectos pueden estar en los programas, en los diseños o incluso en los requisitos, las especificaciones o en otra documentación. Los defectos pueden ser sentencias extra o redundantes, sentencias incorrectas o secciones del programa omitidas. Un defecto, es cualquier cosa que reduce la capacidad de los programas para cumplir completa y efectivamente las necesidades de los usuarios. Un defecto es una cosa objetiva. Es algo que puedes identificar, describir y contabilizar.

Errores sencillos de codificación pueden producir defectos muy destructivos o que sea difícil encontrarlos. A la inversa, muchos defectos sofisticados de diseño pueden encontrarse fácilmente. La sofisticación del error de diseño y el impacto del defecto resultante, son en gran parte independientes. Los errores triviales de implementación pueden causar serios problemas en el sistema. En efecto, la fuente de muchos defectos software son simples descuidos y errores del programador. Aunque los aspectos de diseño son siempre importantes, cuando comienzas a codificar los programas, normalmente tienen pocos defectos de diseño comparados con el número de simples descuidos, erratas y pifias. Para mejorar la calidad del programa, es esencial que los ingenieros aprendan a gestionar todos los defectos que introducen en sus programas.

Es importante separar la cuestión de encontrar o identificar los defectos de la determinación de sus causas. La simple contabilización y registro de los defectos en los productos software no es la especificación de las causas ni la asignación de culpas. Los defectos cometidos, sin embargo, tienen sus causas. Puedes haber cometido un error al escribir el nombre de un parámetro, omitido un signo de puntuación o llamado incorrectamente un procedimiento. Todos estos errores causan defectos. Todos los defectos, por consiguiente, provienen de errores humanos y muchos de los que los ingenieros del software cometen, causan defectos en los programas.

Los errores son cosas incorrectas que cometen las personas y, sin tener en cuenta cuándo y quién los comete, los defectos son elementos defectuosos de los programas. Así, las personas *cometen* errores o equivocaciones mientras que los programas *tienen* defectos. Cuando los ingenieros cometen errores que conducen a defectos, nosotros nos referimos a esto como la introducción de defectos. Esto significa que para reducir el número de defectos que introduces en tus productos, debes cambiar la forma de hacer las cosas. Para eliminar los defectos en tus productos, sin embargo, sencillamente tienes que encontrarlos. La eliminación de defectos es, por lo tanto, un proceso más sencillo que la prevención de defectos. La prevención de defectos es un aspecto importante y prioritario que requiere un estudio comprensivo de todo el proceso de desarrollo del software [Humphrey 89]. El resto de este libro nos centraremos en la eliminación de defectos.

A no ser que los ingenieros encuentren y corrijan los defectos que introducen, estos defectos llegarán al final a los productos acabados. El problema es que lleva mucho tiempo y dinero encontrar y corregir los defectos del software. Para producir pocos defectos, debes aprender de los defectos que has introducido, identificar los errores que los causan y aprender cómo evitar repetir el mismo error en el futuro. Puesto que los productos defectuosos pueden ser caros de probar, difícil de corregir y posiblemente peligrosos de utilizar, es importante que aprendas a minimizar el número de defectos que dejas en tus productos. Este libro muestra cómo hacer esto.

Los defectos deberían ser importantes para cada ingeniero del software no sólo porque afectan a los usuarios, sino también porque más de la mitad del esfuerzo de las organizaciones de software está dedicado a encontrar y corregir los defectos. Puesto que el tiempo de pruebas es difícil de predecir, los defectos son, a menudo, la causa principal de los problemas de costes y programaciones.

12.4

DEFECTOS VERSUS BUGS

Algunas personas, erróneamente se refieren a los defectos del software como bugs'. Cuando lo denominan como bugs, parece que son cosas mal-ditas que deberían ser aplastadas o ignoradas. Esto trivializa un problema crítico y fomenta una actitud errónea. Así, cuando un ingeniero dice que hay solamente unos pocos bugs en el programa, la reacción es de alivio. Supongamos, sin embargo, que los denominamos bombas de efecto retardado en vez de bugs. ¿Sentirías la misma sensación de alivio si un pro-

Es habitual usar el término bug (“gusano”) para denotar los errores que hay en un programa. El autor se refiere a esta forma de referenciar los errores (N. del T.).

gramador te dijese que ha probado a fondo un programa y que ha dejado solamente unas pocas bombas de efecto retardado? Utilizando términos diferentes, tu actitud cambia completamente.

Los defectos son más parecidos a las bombas de efecto retardado que a los bugs. Aunque todos no tendrán un impacto explosivo, alguno si podría tenerlo. Cuando los programas se utilizan mucho y de manera distinta a la que sus diseñadores habían previsto, los errores triviales pueden tener consecuencias imprevisibles. Cuanto más ampliamente se utilizan los sistemas software, más se incrementan las nuevas necesidades a satisfacer, los problemas latentes podrán aparecer y un defecto aparentemente trivial puede ser verdaderamente destructivo.

En este momento, aquellos lectores que han codificado varios programas, probablemente moverán sus cabezas y pensarán que estoy exagerando el caso. En cierto sentido, lo estoy haciendo. La gran mayoría de defectos triviales tienen consecuencias triviales. Desafortunadamente, sin embargo, un pequeño porcentaje de errores aparentemente tontos pueden causar serios problemas. Por ejemplo, un simple error de inicialización causó el desbordamiento de un buffer. Esto originó que un sistema de control de ferrocarril perdiese datos. Entonces, cuando hubo una interrupción, el sistema no pudo restablecerse rápidamente y todos los trenes en varios miles de millas de trayectoria tuvieron que parar durante varias horas, mientras los datos necesarios se volvieron a introducir.

Algún porcentaje de defectos en un programa, probablemente tendrá consecuencias impredecibles. Si supiésemos por adelantado cuáles son dichos errores, entonces los podríamos arreglar y no preocuparnos del resto. Desafortunadamente, no hay forma de hacer esto y cualquier defecto que se ha dejado pasar, puede potencialmente tener serias consecuencias. Aunque es verdad que muchos programas no se utilizan en aplicaciones donde el fallo es más que una molestia, el número está creciendo. Así, aunque los defectos puede que no sean ahora una cuestión importante para ti, pronto lo podrán ser. Es importante que aprendas a gestionar los defectos ahora para que estés preparado cuando verdaderamente necesites hacer programas de gran calidad.

El ingeniero del software que escribe un programa, está más capacitado para encontrar y corregir sus defectos. Es importante que los ingenieros del software asuman la responsabilidad personal con respecto a la calidad de los programas que hacen. Aprender a codificar programas libres de defectos es, sin embargo, un enorme reto. Esto no es algo que cualquiera pueda hacer rápidamente o fácilmente. Requiere información, una técnica efectiva y habilidad. Utilizando los métodos descritos en este libro, puedes desarrollar y perfeccionar tu capacidad para hacer programas de gran calidad. Después de todo, si no te esfuerzas en hacer un trabajo sin defectos, probablemente nunca lo conseguirás.

12.5

TIPOS DE DEFECTOS

Para analizar los defectos, es útil dividirlos en categorías. Este libro clasifica los defectos en 10 tipos generales. Clasificando los defectos en unos pocos tipos, puedes ver rápidamente qué categorías causan mayor problema y así, te puedes centrar en su prevención y eliminación. Esto, naturalmente, es la clave de la gestión de defectos. Céntrate en unos pocos tipos de defectos que son los más problemáticos. Una vez que estos tipos estén bajo control, identifica el conjunto siguiente y trabaja sobre ellos, y así indefinidamente.

Los tipos de defectos utilizados en este libro se muestran en la Tabla 12.1. Esta lista procede del trabajo de Chillarege y sus colegas en el centro de investigación de IBM [Chillarege]. Estudió los defectos en una amplia variedad de productos IBM e identificó sus categorías principales. Estos mismos tipos se han utilizado en el PSP [Humphrey 95].

En la Tabla 12.1, las categorías de defectos se refieren a clases genéricas de problemas. Por ejemplo, los defectos de sintaxis del tipo 20 se refieren a todos aquellos elementos que no coinciden con la especificación del lenguaje de programación. Por ejemplo, omitir un punto y coma, escribir incorrectamente la sentencia `if`, errores de ortografía o declaracio-

Tabla 12.1 Tipos de defectos estándar.

Tipos de defectos		
Número del tipo	Nombre del tipo	Descripción
10	Documentación	comentarios, mensajes
20	Sintaxis	ortografía, puntuación, erratas, formato de las instrucciones
30	Construir, paquetes	gestión del cambio, librerías, control de versión
40	Asignación	declaración, nombres duplicados, ámbito, límites
50	Interfaz	llamadas a procedimientos y referencias, E/S, formatos de usuario
60	Chequeo	mensajes de error, chequeos inadecuados
70	Datos	estructura, contenido
80	Función	lógica, punteros, bucles, recursión, computación, defectos de la función
90	Sistema	configuración, temporización, memoria
100	Entorno	diseño, compilación, pruebas y otros problemas que soporta el sistema

nes inadecuadas. Un defecto de sintaxis del tipo 20 es cualquier defecto que da lugar a un programa sintácticamente incorrecto, sin tener en cuenta como se originaron los defectos o se encontraron. Otra categoría de defecto podría ser la del tipo 80 (el tipo función). Un ejemplo aquí sería el bucle `do-while` con una incorrección lógica en la condición `while`.

La clasificación de la Tabla 12.1, está ordenada por el grado de sofisticación general de la probable causa del defecto. Por ejemplo, los defectos de los tipos 10, 20 y 30 resultan de un simple descuido o error. Los defectos de los tipos 80, 90 y 100, sin embargo, normalmente implican un diseño más sofisticado o cuestiones del sistema.

Antes de refinar cada una de estas 10 categorías de defectos del PSP en subcategorías, espera hasta que hayas reunido los datos de defectos de un número de programas. Entonces puedes ver, dónde un mayor detalle te ayudaría y qué información específica adicional sería necesaria. Por ejemplo, podrías dividir los defectos de sintaxis del tipo 20 en los subtipos 21 para los puntos y comas, el 22 para otros errores de puntuación, el 23 para los problemas de expresiones booleanas, el 24 para formatos de instrucciones incorrectas y así sucesivamente. Antes de definir nuevas subcategorías para cada tipo de defecto, espera hasta que hayas reunido los datos de al menos 100 o más defectos. Aún así, probablemente encontrarás que estos tipos de datos de defectos iniciales son adecuados.

12.6

LA COMPRENSIÓN DE LOS DEFECTOS

El primer paso para gestionar los defectos es entenderlos. Para hacer eso, debes reunir los datos de defectos. Entonces, podrás entender estos errores y comprender cómo evitarlos. Puedes también comprender cómo encontrarlos mejor, corregirlos o prevenir el defecto que todavía introduces.

Para reunir datos de defectos de tus programas, haz lo siguiente:

- Registra cada defecto que encuentres en un programa.
- Registra la información suficiente sobre cada defecto para que puedas entenderlo posteriormente.
- Analiza estos datos para ver qué tipos de defectos causan los mayores problemas.
- Idea formas de encontrar y corregir estos defectos.

Los defectos que introduces y encuentras en tus propios programas, son solamente parte de la historia. Algún día, necesitarás aprender sobre los defectos que otras personas encuentran en tus programas. Puesto que estos defectos se escaparán a todos los esfuerzos de detección y prevención de defectos, serán más importantes para entender y tratar las debilidades de tus procesos personales. Estos defectos se denominan escapes, porque han

escapado a todos tus esfuerzos de eliminación de defectos. Conforme mejore tu proceso personal, los defectos que se escapan serán la última fuente principal de datos para tu mejora personal.

12.7

EL CUADERNO DE REGISTRO DE DEFECTOS

El cuaderno de registro de defectos está diseñado para ayudarte a reunir datos de defectos. El cuaderno se muestra en la Tabla 12.2 y sus instrucciones se indican en la Tabla 12.3. Utiliza este cuaderno para reunir datos de defectos para cada programa que codifiques. Describe cada defecto con bastante detalle para que puedas entenderlo más adelante. Después de haber terminado cada programa, analiza los datos para ver dónde has introducido y eliminado los defectos y qué tipos de defectos causan los principales problemas. Antes de utilizar este cuaderno, lee el resto de ese capítulo y las instrucciones de la Tabla 12.3. Los siguientes párrafos utilizan el ejemplo del Cuaderno de Registro de Defectos de la Tabla 12.4 para mostrar como completar el cuaderno:

- 1. Cuando comiences a desarrollar un programa,** coge varias páginas del Cuaderno de Registro de Defectos y rellena los datos de la cabecera de la primera página. Después de utilizar todos los espacios en la primera página, completa la cabecera antes de comenzar la segunda página.
- 2. Cuando encuentres un defecto por primera vez,** anota su número en el cuaderno, pero no introduzcas el resto de datos hasta que hayas corregido el defecto. Cuando el Estudiante X intentó compilar el programa 10, el compilador mostró más de una docena de mensajes de error. Aunque al principio no sabía qué problema tenía, al menos sabía que era un error. Anotó la fecha y puso un 1 en la casilla Número de la primera línea del cuaderno de defectos. Esto fue para el primer defecto del programa 10. Estos números te ayudarán posteriormente a analizar los datos de los defectos. En programas más grandes, los números de defecto se utilizan para controlar los problemas con correcciones incorrectas y ayudar a la prevención de defectos.
- 3. Utiliza una línea separada para cada defecto.** No agrupes múltiples defectos idénticos en la misma línea.
- 4. Escribe la fecha de localización del defecto.** Si encuentras varios defectos el mismo día, es aceptable dejar las siguientes casillas de la fecha en blanco, hasta la primera anotación del día siguiente. En la Tabla 12.4, el Estudiante X encontró todos los defectos el día 28 de octubre, por lo que no necesitó volver a anotar la fecha, pues supuso que se repetía hasta que no la cambiase.

Tipos de defectos		
10 Documentación	50 Interfaz	90 Sistema
20 Sintaxis	60 Comprobación	100 Entorno
30 Construcción Paquetes	70 Datos	
40 Asignación	80 Función	

Tabla 12.2 Cuaderno de registro de defectos.

Estudiante _____ Fecha _____
 Profesor _____ Programa # _____

Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
_____	_____	_____	_____	_____	_____	_____

Descripción: _____

Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
_____	_____	_____	_____	_____	_____	_____

Descripción: _____

Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
_____	_____	_____	_____	_____	_____	_____

Descripción: _____

Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
_____	_____	_____	_____	_____	_____	_____

Descripción: _____

Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
_____	_____	_____	_____	_____	_____	_____

Descripción: _____

Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
_____	_____	_____	_____	_____	_____	_____

Descripción: _____

Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
_____	_____	_____	_____	_____	_____	_____

Descripción: _____

Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
_____	_____	_____	_____	_____	_____	_____

Descripción: _____

Tabla 12.3 Instrucciones para el cuaderno de registro de defectos.

Propósito	Utiliza esta tabla para mantener los datos de cada defecto que encuentres y corrijas. Utiliza estos datos para completar el Resumen del Plan del Proyecto.
Método	Anota todas las revisiones, compilaciones y pruebas de defectos en este cuaderno. Anota cada defecto de forma separada y completa. Si necesitas espacio adicional, utiliza otra copia de la tabla.
Cabecera	Introduce los siguientes datos: <ul style="list-style-type: none">• Tu nombre.• Fecha actual.• Nombre del profesor.• Número de programa.
Fecha	Anota la fecha en la que se encontró el defecto.
Número	Número de cada defecto. Para cada programa, utiliza una numeración secuencia, comenzando por el 1 (o 001, etc.).
Tipo	Anota el tipo de defecto, según la lista de tipos de defectos de la Tabla 12.1 (también resumida en la parte superior izquierda del cuaderno de defectos). Utiliza tu criterio para seleccionar que tipo aplicar.
Introducido	Anota la fase en la que se introdujo el defecto. Utiliza tu criterio.
Eliminado	Anota la fecha en la que se eliminó el defecto. Generalmente, ésta sería la fase durante la cual encontraste y corregiste el defecto.
Tiempo de corrección	Estima o mide el tiempo necesario para encontrar y corregir el defecto. Puedes utilizar un cronómetro si lo deseas.
Defecto corregido	Puedes ignorar esta casilla la primera vez. Si introduces este defecto mientras estás arreglando otro, anota el número del defecto incorrectamente corregido. Si no puedes identificar el número de defecto, anota una X en la casilla de Defecto corregido.
Descripción	Escribe una breve descripción del defecto. Haz la descripción lo suficientemente clara para que recuerdes posteriormente, el error que causó el defecto y por qué lo hiciste.

5. Despues de corregir el defecto, anota el tipo de defecto. Aunque puedas confundirte sobre qué tipo es el adecuado, utiliza tu mejor criterio. No dediques mucho tiempo preocupándote sobre qué tipo de defecto es el más preciso. Sin embargo, intenta ser razo-nablemente coherente. Sobre el defecto 1 en la Tabla 12.4, por ejemplo, el Estudiante X encontró que el problema era un punto

Tipos de defectos			
10 Documentación	50 Interfaz	90 Sistema	
20 Sintaxis	60 Comprobación	100 Entorno	
30 Construcción Paquetes	70 Datos		
40 Asignación	80 Función		

Tabla 12.4 Ejemplo del cuaderno de registro de defectos.

y coma olvidado. Una vez resuelto el problema, anotó el número 20 para el defecto 1 en la casilla de Tipo.

6. *Anota la fase del proceso en la que introdujiste el defecto.* Aunque esto pueda no estar siempre claro, no debería ser un problema para programas pequeños. Utiliza tu mejor criterio y no te preocupes mucho tiempo de este tema. En el ejemplo, el Estudiante X estaba convencido de que había cometido el error del punto y coma cuando estaba codificando el programa, por eso puso la palabra *codificar* en la casilla de Introducido.
7. *Anota la fase del proceso cuando hayas eliminado el defecto.* Esta es normalmente la fase en la que encuentras el defecto. Después de iniciar la fase de *compilación*, por ejemplo, anota la palabra *compilar* para la fase de eliminación. Aquí, para el defecto 1, el Estudiante X estaba en la fase de compilación cuando encontró y corrigió el defecto, por eso anotó la palabra *compilar* en la casilla de Eliminado.
8. *Para el tiempo de corrección del defecto, estima el tiempo en que te diste cuenta* y comenzaste a trabajar sobre el defecto hasta que lo acabaste de corregir y chequear. Cuando comenzó a corregir el defecto 1, el Estudiante X anotó la hora de su reloj. Una vez que había arreglado el problema y comprobado para asegurarse de que estaba correctamente corregido, de nuevo comprobó su reloj y vio que solamente le había dedicado un minuto. Generalmente, para defectos de compilación, el tiempo de corrección será solamente de un minuto aproximadamente. Para los defectos encontrados en las pruebas, sin embargo, la corrección puede llevar mucho más tiempo. Podrías utilizar un reloj o un cronómetro para medir el tiempo de corrección, pero para correcciones pequeñas, tu criterio será adecuado.
9. *La casilla de Defectos Corregidos* es para los defectos introducidos mientras corriges otros defectos. Aunque esto será importante más adelante, ignóralo por ahora.
10. *Escribe una breve descripción del defecto en la sección de descripción.* Haz esto tan breve y sencillo como sea posible, pero describe el defecto claramente. Por ejemplo, simplemente anota un “;” para designar un punto y coma omitido. Para un defecto lógico más sofisticado, escribe varias líneas, escribe en las siguientes líneas del cuaderno de defectos si es necesario. Para el defecto 1, el Estudiante X simplemente anotó “omitido ;”. Para muchos de los defectos de la Tabla 12.4, tuvo que poner una descripción más detallada. Puesto que estas descripciones son únicamente para tu uso, no es necesario escribir más de lo preciso para que puedas recordar el problema.

A menudo, las personas se confunden sobre los tipos de defectos y piensan que deberían tener un tipo especial para interpretaciones erróneas y confusiones. Por ejemplo, si no entendiste los requisitos o no estabas familiarizado con el entorno de desarrollo, probablemente cometiste muchos errores. Esta cuestión es importante, pero está relacionada con las *causas* del defecto. Por lo que al *tipo* de defecto se refiere, hay solamente dos cuestiones. ¿Había algo erróneo en el producto? y si es así, ¿cuál era el tipo de defecto del producto? Así, aunque entender la causa es necesario para prevenir los defectos, el tipo de defecto solamente describe lo que estaba incorrecto en el producto.

12.8

LA CONTABILIZACIÓN DE LOS DEFECTOS

Aunque la definición de un defecto puede parecer obvia, no lo es. Durante la compilación, por ejemplo, cuenta solamente los cambios que haces. Es decir, si el compilador presenta 10 mensajes de error por una omisión del punto y coma, la omisión del punto y coma es un único defecto. Así, anota un defecto en el Cuaderno de Registro de Defectos para cada corrección del programa, sin tener en cuenta la naturaleza de la corrección y el número de mensajes de error del compilador.

De forma similar, cuando encuentres un defecto de diseño mientras estés codificando, se considerará un defecto de diseño. Mientras diseñas, sin embargo, con frecuencia puedes cambiar tu idea de cómo hacer algo. Si estás corrigiendo un error en los requisitos o en las especificaciones, eso sería un defecto de requisitos o de especificación. Si, por el contrario, has pensado una forma mejor de hacer el diseño, no sería un defecto. A menudo, advertirás y corregirás errores conforme los vas cometiendo. Dichos ajustes son las cosas más naturales de un pensamiento creativo y no son defectos. La clave está en registrar aquellos defectos que has dejado en el producto cuando hayas acabado el diseño inicial o terminado la codificación.

Por ejemplo, si escribes una línea de código e inmediatamente ves un error en el nombre del parámetro y lo corriges, este error no es un defecto. Si, por el contrario, acabas de codificar el programa y posteriormente observas el error, entonces sí sería un defecto y lo contabilizarías. Así, si normalmente compruebas la corrección de cada línea después de introducirla, los defectos que encuentres de esta forma no es necesario contabilizarlos.

Comienza a contabilizar los defectos cuando termines una fase de un producto o parte del mismo. Después de la fase de diseño, por ejemplo, contarías todos los defectos de diseño. Supongamos, sin embargo, que estás codificando dos procedimientos de un programa. Después de codificar

el primero, decides codificar el segundo, antes de comenzar la compilación del primero. A mitad de codificar el segundo procedimiento, te das cuenta de que has dado un nombre equivocado a un parámetro en el primer procedimiento. Esto es un defecto, porque aunque estés en la fase de codificación, en ese momento habías terminado la codificación del primer procedimiento.

Observa que en este libro no se te exige contabilizar los defectos encontrados durante las fases de diseño y codificación. Inicialmente, es importante concentrarte sobre aquellos defectos encontrados durante la compilación y pruebas. Una vez que estés acostumbrado a reunir datos de defectos, sabrás mejor por qué son necesarios dichos datos. Entonces puedes querer aprender más sobre los errores que cometes y corriges durante las fases de codificación y diseño. Puesto que probablemente cometerás muchos errores mientras diseñas y codificas, estas son las fases donde debes tratar de entender las causas de los defectos y ver cómo prevenirlos. Por el momento, sin embargo, comienza con aquellos defectos que encuentres en la compilación y en las pruebas.

12.9 LA UTILIZACIÓN DEL CUADERNO DE REGISTRO DE DEFECTOS

¿Por qué deberías contar los defectos? Conforme reúnas datos de defectos, recuerda por qué lo estás haciendo:

- *Para mejorar tu programación.* Estos datos de los defectos te ayudan a mejorar la forma que tienes de escribir programas. Aunque es fácil defenderte frente a los defectos, no puedes gestionar los defectos si no los entiendes. Esto significa que debes reunir datos exactos sobre ellos. Si pones excusas o pretendes, por ejemplo, que los errores sintácticos no se cuenten si son detectados por un corrector sintáctico en vez de por un compilador, te engañarás a ti mismo. Si estás deseando engañarte a ti mismo, no esperes mejorar.
- *Para reducir el número de defectos en tus programas.* Todos introducen defectos pero, utilizando métodos cuidadosos y adecuados, puedes reducir el número de defectos introducidos.
- *Para ahorrar tiempo.* Los errores causan más errores. Los grandes defectos permanecen en un programa, debido al tiempo que dedicas a encontrarlos y lo difícil que es corregirlos. Los problemas de requisitos conducen a diseños inadecuados. Los errores de diseño causan errores de implementación. Los errores de implementación introducen defectos en los programas. Por ello, es importante eliminar los defectos lo antes posible después de introducirlos.

- *Para ahorrar dinero.* Los defectos son caros. Después de una prueba, los costes de encontrar y corregir los defectos aumentan en un factor de 10 para cada subsiguiente fase de pruebas o de mantenimiento.
- *Paru hacer tu trabajo de una forma responsable.* Los defectos son introducidos por los ingenieros y es su responsabilidad encontrarlos y corregirlos.

12.10 EL PROCESO DEL PSP ACTUALIZADO

La guía del proceso del PSP actualizada se muestra en la Tabla 12.5. El añadido principal es la reunión y registro de datos de defectos. La tabla del Resumen del Plan del Proyecto se muestra en la Tabla 12.6 y sus instrucciones están en la Tabla 12.7. Observa que no hay espacio para registrar los defectos introducidos durante la fase postmortem. Aunque no es probable que encuentres o introduzcas defectos durante el análisis postmortem, es posible. Un ejemplo podría ser que observases un defecto mientras contabilizas las LOC Nuevas & Cambiadas. Puedes introducir un defecto durante la fase postmortem si cometes un error mientras corriges el primer defecto. Los párrafos siguientes explican cómo se completan las partes nuevas del Resumen del Plan del Proyecto de la Tabla 12.8.

Durante la fase postmortem, revisa el Cuaderno de Registro de Defectos y contabiliza el número de defectos introducidos en cada fase. En el Cuaderno de Registro de Defectos de la Tabla 12.4, el Estudiante X, en primer lugar contabilizó los defectos 3 y 6 como introducidos durante la fase de diseño, por lo que anotó un 2 en la casilla Real de la fila de Diseño en la Tabla 12.8. Los otros 6 defectos fueron todos introducidos durante la codificación, por ello anotó un 6 en la fila de codificación. Resultando un total de 8 defectos. Aunque introdujeras la mayor parte de los defectos en la fase de codificación, puedes introducir unos pocos en las fases de diseño, compilación o pruebas. Ocasionalmente, con programas más complejos, puedes introducir algún defecto durante la fase de planificación.

A continuación, contabiliza los defectos eliminados en cada fase. El Estudiante X contabilizó 6 defectos eliminados en la fase de compilación y dos en la de pruebas, por lo que anotó un 6 y un 2 en las dos filas de la sección de eliminados. De nuevo, obtendremos un total de 8. Con el PSP, probablemente comenzarás encontrando la mayor parte de los defectos durante la fase de compilación. Las pruebas, sin embargo, necesitan más tiempo, puesto que es más difícil encontrar y corregir defectos en dicha fase.

Después de anotar el número de defectos introducidos y eliminados, completa las columnas de Hasta la Fecha y % Hasta la Fecha de la misma forma que rellenes las mismas columnas con los datos de tiempos (véase la Sección 11.7 del Capítulo 11). No necesitarás los datos de defectos

Tabla 12.5 Guión del proceso del PSP.

	Propósito	Guiaerte en el desarrollo de pequeños programas.
	Criterios de entrada	La descripción del problema. Tabla Resumen del Plan del Proyecto del PSP. Datos de tamaños y tiempos reales para programas anteriores. Cuaderno de Registro de Tiempos. Cuaderno de Registro de Defectos.
1	Planificación	Obtén una descripción de las funciones del programa. Estima las LOC Máx., Mín. y total requeridas. Determina los Minutos/LOC. Calcula los tiempos de desarrollo Máx., Mín., y total. Anota los datos del plan en la tabla Resumen del Plan del Proyecto. Anota el tiempo de planificación en el Cuaderno de Registro de Tiempos.
2	Diseño	Diseña el programa. Anota el diseño en el formato especificado. Anota el tiempo de diseño en el Cuaderno de Registro de Tiempos.
3	Codificación	Implementa el diseño. Utiliza un formato estándar para introducir el código. Anota el tiempo de codificación en el Cuaderno de Registro de Tiempos.
4	Compilación	Compila el programa. Corrige y anota todos los defectos encontrados. Anota el tiempo de compilación en el Cuaderno de Registro de Tiempos.
5	Pruebas	Prueba el programa. Corrige y anota todos los defectos encontrados. Anota el tiempo de pruebas en el Cuaderno de Registro de Tiempos.
6	Postmortem	Completa la tabla de Resumen del Plan del Proyecto con los datos de tiempo, tamaño y defectos reales. Anota el tiempo postmortem en el Cuaderno de Registro de Tiempos.
	Criterios de salida	Programa probado a fondo. Diseño adecuadamente documentado. Listado completo del programa. Resumen del Plan del Proyecto completado. Cuadernos de Tiempos y defectos completados.

Hasta la Fecha y % Hasta la Fecha hasta el Capítulo 15, que será cuando comenzarás a estimar el número de defectos introducidos y eliminados.

Con los datos del % Hasta la Fecha, es sorprendente la precisión con que los ingenieros pueden estimar el número de defectos que introducen y eliminan. Las personas son animales de costumbres y nuestras costum-

Tabla 12.6 Resumen del plan del proyecto del PSP.

Estudiante _____	Fecha _____		
Programa _____	Programa# _____		
Profesor _____	Lenguaje _____		
<i>Resumen</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la Fecha</i>
Minutos/LOC			
LOC/Hora			
Defectos/KLOC			
Rendimiento			
VIF			
<i>Tamaño Programa (LOC):</i>			
Total Nuevo & Cambiado			
Tamaño Máximo			
Tamaño Mínimo			
<i>Tiempo por Fase (min.)</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la Fecha</i>
Planificación			
Diseño			
Codificación			
<i>Revisión del código</i>			
Compilación			
Pruebas			
Postmorten			
Total			
Tiempo Máximo			
Tiempo Mínimo			
<i>Defectos introducidos</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la Fecha</i>
Planificación			
Diseño			
Codificación			
<i>Revisión del código</i>			
Compilación			
Pruebas			
Total			
<i>Defectos eliminados</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la Fecha</i>
Planificación			
Diseño			
Codificación			
<i>Revisión del código</i>			
Compilación			
Pruebas			
Total			

bres gobiernan nuestros errores. Mientras no cambiamos estos hábitos, continuaremos cometiendo errores similares. Así, a no ser que hagas un gran cambio, tal como utilizar un proceso diferente, trabajar en aplicaciones más complejas o modificar el entorno de desarrollo, probablemente introducirás aproximadamente el mismo número de defectos en el próximo programa de la misma forma que lo hiciste la Última vez.

El resto del Resumen del Plan del Proyecto se completa de la misma forma como hemos visto anteriormente. Hay, sin embargo, unas pocas

Tabla 12.7 Instrucciones del resumen del plan del proyecto del PSP.

Propósito	Esta tabla trata los datos estimados y reales de los proyectos de una forma cómoda y fácilmente recuperable.
Cabecera	Introduce los siguientes datos: <ul style="list-style-type: none"> • Tu nombre y fecha de hoy. • Nombre y número de programa. • Nombre del profesor. • El lenguaje que utilizarás para escribir el programa.
Minutos/LOC	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> • Escribe los Minutos/LOC planificados para este proyecto. Utiliza la velocidad Hasta la Fecha de un programa reciente del Cuaderno de Trabajos o del Resumen del Plan del Proyecto. Después del desarrollo: <ul style="list-style-type: none"> • Divide el tiempo total de desarrollo por el tamaño real del programa para obtener los Minutos/LOC reales y los Minutos/LOC Hasta la Fecha. • Por ejemplo, si el proyecto se hizo en 196 minutos e hiciste 29 LOC, los Minutos/LOC serían $196/29 = 6,76$.
LOC/Hora	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> • Calcula las LOC por hora planificada para este programa dividiendo 60 por los Minutos/LOC de la casilla de Plan. Después del desarrollo: <ul style="list-style-type: none"> • Para LOC/Hora Real y Hasta la Fecha divide 60 por Minutos/LOC Reales y Hasta la Fecha. • Para los 6,76 Minutos/LOC Reales, tenemos $60/6,76 = 8,88$ LOC/Hora Reales.
Tamaño Programa (LOC)	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> • Escribe bajo la columna plan, el valor estimado de Total Nuevas & Cambiadas, Máximo y Mínimo. Después del desarrollo: <ul style="list-style-type: none"> • Cuenta y escribe las LOC Nuevas & Cambiadas Reales. • Para la columna Hasta la Fecha, añade LOC reales Nuevas & Cambiadas a las LOC Hasta la Fecha Nuevas & Cambiadas de programas anteriores.
Tiempo por fase Plan	Para el tiempo total de desarrollo, multiplica el valor de las LOC Total Nuevas & Cambiadas por Minutos/LOC. Para el tiempo Máximo, multiplica el tamaño Máximo por los Minutos/LOC. Para el tiempo Mínimo, multiplica el tamaño Mínimo por los Minutos/LOC. Del Resumen del Plan del Proyecto de un programa reciente, busca los valores de % Hasta la Fecha para cada fase. Utilizando el % Hasta la Fecha de programas anteriores calcula el tiempo planificado para cada fase.
Real	Una vez acabado el trabajo, anota el tiempo real en minutos que has gastado en cada fase del desarrollo. Obtén estos datos del Cuaderno de Tiempos.

(Continúa)

Tabla 12.7 Instrucciones del resumen del plan del proyecto del PSP.
(Continuación)

Hasta la Fecha	Para cada fase, escribe la suma del tiempo real y el tiempo Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, escribe 100 multiplicado por el tiempo Hasta la Fecha y lo divides por el total del tiempo Hasta la Fecha.
Defectos reales introducidos	<i>Después del desarrollo, localiza y anota el número real de defectos introducidos en cada fase.</i>
Hasta la Fecha	Para cada fase, escribe la suma de los defectos reales y los defectos Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, multiplica por 100 los Defectos Hasta la Fecha para esa fase y divídelos por el total de defectos Hasta la Fecha.
Defectos reales eliminados	<i>Después del desarrollo, localiza y anota el número real de defectos eliminados en cada fase.</i>
Hasta la Fecha	Para cada fase, escribe la suma de los defectos reales y los defectos Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, multiplica por 100 los Defectos Hasta la Fecha para esa fase y divídelos por el total de defectos Hasta la Fecha.

consideraciones a tener en cuenta: primero, el dato Hasta la Fecha en Minutos/LOC se obtiene del cociente entre el tiempo total de desarrollo Hasta la Fecha y las LOC Nuevas & Cambiadas Hasta la Fecha; tenemos, $682/105=6,50$ Minutos/LOC. Segundo, las LOC/Hora se calculan dividiendo 60 por los Minutos/LOC hasta la Fecha, tenemos $60/6,50=9,23$ LOC/Hora. Observa que con estas velocidades de Hasta la Fecha, no necesitas controlar las Unidades y Medias del Cuaderno de Trabajos de los programas desarrollados. Este cuaderno es una referencia muy adecuada para la información del proyecto, sin embargo, te sugiero que continúes revisando los Cuadernos de Trabajo del programa. Si tus experiencias son como las mías, rápidamente acumularás una gran cantidad de datos. Yo he controlado hasta ahora unos 200 trabajos, y he almacenado mis datos de proyectos por número de trabajo. Así, encuentro que el Cuaderno de Trabajos es la forma más adecuada para identificar datos históricos o encontrar datos de un proyecto particular.

RESUMEN

La calidad del software consiste en satisfacer las necesidades de los usuarios haciendo el trabajo de los mismos de una forma fiable y consistente. Esto requiere que el software que hagas tenga pocos defectos.

Tabla 12.8 Ejemplo del resumen del plan del proyecto del PSP.

Estudiante	Estudiante X	Fecha	28/10/96
Programa		Programa#	10
Profesor	Sr. Z	Lenguaje	Ada
Resumen	Plan	Real	Hasta la Fecha
Minutos/LOC	6,76	6,12	6,50
LOC/Hora	8,88	9,80	9,23
Defectos/KLOC			
Rendimiento			
VIF			
Tamaño Programa (LOC):			
Total Nuevo & Cambiado	44	57	105
Tamaño Máximo	58		
Tamaño Mínimo	30		
Tiempo por Fase (min.)	Plan	Real	Hasta la Fecha % Hasta la Fecha
Planificación	13	18	33 4,8
Diseño	11	43	55 8,1
Codificación	130	162	300 45,2
Revisión del código			
Compilación	44	2	70 10,2
Pruebas	82	73	165 24,2
Postmorten	17	32	51 7,5
Total	297	349	682 100,0
Tiempo Máximo	392		
Tiempo Mínimo	203		
Defectos introducidos	Plan	Actual	Hasta la Fecha % Hasta la Fecha Def./Hora
Planificación			
Diseño		2	2 25,0
Codificación		6	6 75,0
Revisión del código			
Compilación			
Pruebas			
Total		8	8 100,0
Defectos eliminados	Plan	Actual	Hasta la Fecha % Hasta la Fecha Def./Hora
Planificación			
Diseño			
Codificación			
Revisión del código			
Compilación		6	6 75,0
Pruebas		2	2 25,0
Total		8	8 100,0

Un defecto del software es algo en el producto que está incorrecto. Los defectos son causados por errores humanos. Puesto que los defectos son costosos de encontrar y corregir, es muy eficiente que los ingenieros encuentren y corrijan inmediatamente los defectos introducidos.

El primer paso en la gestión de defectos es entenderlos. Para esto, el programador necesita reunir datos de defectos, analizar estos datos y determinar la mejor forma de prevenir, localizar y corregir dichos defectos.

El Estándar de Tipos de Defectos es un estándar simplificado utilizado en este libro para ayudarte a identificar tus categorías de defectos más importantes. Después de que tengas algunos datos de defectos, puedes desear extender estas categorías. Utiliza el Cuaderno de Defectos para reunir los datos de los defectos. Registra cada defecto utilizando una casilla del cuaderno para cada defecto. Para cada programa, resume estos datos en la tabla del Plan del Proyecto.

EJERCICIO 12

Utiliza el Cuaderno de Registro de Defectos para registrar cada defecto que encuentres en los programas que escribas. Identifica el programa donde encuentras los defectos, anota cada defecto como un registro separado y describe detalladamente cada defecto. Resume los datos de defectos en el Resumen del Plan del Proyecto para cada programa.

También, entrega una copia del Cuaderno de Tiempos, del Cuaderno de Trabajos y del Resumen Semanal de Actividades que no hayas entregado todavía, y entrega una planificación y un Cuaderno de Registro de Defectos para cada nuevo programa que desarrolles.

REFERENCIAS

[Chillarege] Chillarege, Ram, Inderpal S. Bhandari, Jarir K. Chaar, Michael J. Halliday, Diane S. Moebus, Bonnie K. Ray, and Man-Yuen Wong. "Orthogonal Defect Classification-A Concept for In-Process Measurements." *IEEE Transactions on Software Engineering*, vol., 18, no. 11, Nov 1992, pp. 943-956.

[Humphrey 89] Humphrey, W.S. *Managing the Software Process*. Reading, MA: Addison-Wesley, 1989.

[Humphrey 95] Humphrey, W.S. *A Discipline for Software Engineering*. Reading, MA: Addison-Wesley, 1995.

CAPITULO 13

Encontrar defectos

Hay varias formas de encontrar defectos. Este capítulo resume brevemente las principales alternativas y muestra cómo un método particular, la revisión de código, puede ayudar a mejorar la productividad y calidad de tu trabajo. En el ejercicio, harás revisiones de código para los siguientes programas que escribas.

13.1 UN COMPROMISO PERSONAL CON LA CALIDAD

A pesar de todas las herramientas y métodos disponibles, el factor más importante en la calidad de un programa es el compromiso personal del ingeniero del software a desarrollar un producto de calidad. Cuando los ingenieros se comprometen con la calidad, tienen más cuidado con su trabajo y se enorgullecen de la calidad de los productos que hacen. El PSP puede ayudarte a hacer productos de calidad, mostrándote cómo utilizar métodos efectivos de calidad. El primer paso, es entender los defectos que has introducido en programas anteriores. Entonces, utilizando los métodos del PSP, aprender rápidamente a encontrar y corregir los defectos. Este y los siguientes capítulos describen cómo hacerlo.

13.2 LOS PASOS PARA ENCONTRAR DEFECTOS

Aunque no hay forma de acabar con la introducción de defectos, es posible encontrar y eliminar casi todos los defectos al principio del desarro-

llo. Despues de que hayas aprendido el PSP, encontrarás que eliminar los defectos al principio, te ahorrará tiempo y harás mejores productos. Por ejemplo, si pudieras encontrar y corregir un defecto de diseño antes de hacer el código, no gastarías tiempo implementando un diseño incorrecto. De forma similar, cuando corriges defectos de codificación antes de compilar y hacer las pruebas, ahorras tiempo que tendrías que dedicar a encontrar y corregir estos defectos durante la fase de compilación y pruebas. Este capítulo muestra cómo encontrar los defectos y proporciona datos que puedes utilizar para evaluar la efectividad de estos métodos de eliminación de defectos.

Hay varias formas de encontrar los defectos en un programa. En esencia, todos estos métodos implican los siguientes pasos:

1. Identificar los síntomas del defecto.
2. Deducir de estos síntomas la localización del defecto.
3. Entender lo que es erróneo en el programa.
4. Decidir cómo corregir el defecto.
5. Hacer la corrección.
6. Verificar que el arreglo ha resuelto el problema.

13.3

FORMAS DE ENCONTRAR Y CORREGIR DEFECTOS

Se han inventado varias herramientas y ayudas para ayudar a los ingenieros en estos pasos. La primera herramienta que los ingenieros normalmente utilizan es un compilador. Para entender cómo y por qué un compilador ayuda a encontrar los defectos, es importante discutir su propósito. Fundamentalmente, el trabajo del compilador es generar código. Así, un compilador explorará todo el código fuente para ver si puede generar código. Si puede, lo hará, tanto si el código es correcto como si no.

Así, el compilador generará código hasta que encuentre algunos caracteres que no pueda interpretar. Por ejemplo, si pones la cadena de caracteres ABC en un programa fuente y no la habías declarado, el compilador marcará esta cadena como un error. Los compiladores pueden identificar muchos defectos sintácticos, pero no te pueden decir lo que pretendes. Así, los compiladores, a menudo, proporcionan muchos mensajes de error para defectos aparentemente sencillos. Los compiladores, sin embargo, solamente proporcionan síntomas de defectos y debes entender dónde y cuál es el problema. Aunque normalmente harás esto rápidamente, en ocasiones puedes necesitar mucha dedicación.

Los compiladores no detectarán cada error tipográfico, de puntuación u otro defecto sintáctico. La razón es porque los compiladores, a menudo,

pueden generar código de programas fuente defectuosos. Aunque muchos de estos defectos que pasan inadvertidos provienen de diseños inadecuados, algunos podrían ser simples errores sintácticos. Puede parecer improbable que un compilador pudiese pasar por alto errores sintácticos, pero mis datos de varios miles de defectos de C++ muestran que esto sucedió en el 9,4% de los errores sintácticos que cometí. Así como un corrector ortográfico no puede detectar todos los errores ortográficos, el compilador no detectará todos los defectos sintácticos.

Una segunda forma de encontrar defectos, es por medio de las pruebas. Aunque hay muchas clases de pruebas, todas requieren que los examinadores proporcionen datos de prueba y condiciones de prueba (algunas veces llamadas casos de prueba o escenarios de prueba). La calidad de las pruebas está gobernada por el grado en que estos escenarios cubren todas las funciones importantes del programa. El examinador, entonces, ejecuta estos casos de prueba para ver si el programa proporciona los resultados adecuados. Esto implica otra responsabilidad del examinador: comprender que los resultados de estas pruebas deberían parecerse si el programa trabajase correctamente.

Aunque las pruebas pueden utilizarse para comprobar casi cualquier función del programa, tienen varias desventajas. Primero, como con los compiladores, las pruebas solo suponen el primer paso de corrección de defectos. Es decir, aún tienes que moverte desde los síntomas a los problemas antes de comenzar a trabajar en la corrección. Otro problema, es que cada prueba verifica solamente un conjunto de condiciones del programa. Es decir, si el programa multiplica dos números, x e y , y lo pruebas con $x=1$ e $y=18$, sabrías solamente que funciona para esos valores. No sabrías, por ejemplo, cómo trabaja el programa con números negativos, o con el cero, o con números positivos o negativos muy grandes en el sistema numérico, o con cualquier otro par de números. Para comprobar todas estas posibilidades tendrías que hacer muchas pruebas. Puesto que cada programa sencillo implica muchas combinaciones posibles de datos y condiciones operativas, unas pruebas globales consumen tiempo. En efecto, para cualquier programa sencillo, una prueba global es prácticamente imposible.

La tercera forma de encontrar los defectos, es la más común de todas. Consiste en entregar programas defectuosos y esperar que los usuarios identifiquen e informen de los defectos. Esta es la estrategia más costosa. Por ejemplo, durante un año, IBM gastó unos 250 millones de dólares en reparar y reinstalar correcciones de los 13.000 defectos detectados por los clientes. Esto supone unos 20.000 dólares por defecto.

Por Último, indicar que la forma más efectiva de encontrar y corregir defectos es revisar personalmente el código fuente del programa. Aunque esto puede parecer una forma difícil de limpiar un programa de-

fectuoso, se trata de la forma más rápida y eficiente. Este capítulo explica el porqué.

13.4

LA REVISIÓN DEL CÓDIGO

Una revisión de código es una forma de encontrar defectos rápidamente. Para hacer una revisión de código, estudias el código fuente para encontrar los errores. Esto es mejor hacerlo después de escribir el código fuente y antes de comenzar a compilarlo o probarlo. Puesto que muchos defectos software provienen de simples descuidos y bobadas, son fáciles de encontrar después de hacer el diseño o el código. Es cuando más probablemente, recordarás cuál era tu intención y sabrás cómo corregir cualquier problema.

Aunque hay muchas formas de hacer una revisión de código, la aproximación más común es imprimir un listado con el código fuente y revisarlo línea por línea. Podrías revisar el código en la pantalla del ordenador, pero normalmente, los ingenieros realizan más cómoda la revisión, incluso de los programas pequeños, cuando los tienen en un listado impreso. Los listados también nos permiten movernos rápidamente entre los segmentos de código, tomar notas, o examinar secciones completas.

Aunque la revisión de código consume tiempo, es mucho más eficiente que las pruebas. Los datos de los estudiantes e ingenieros muestran que, la revisión de código es entre 3 y 5 veces más eficiente que ejecutar las pruebas de unidad. Un ingeniero, por ejemplo, encontrará solamente entre 2 a 4 defectos en una hora de pruebas, pero encontrará de 6 a 10 defectos en cada hora de revisión de código.

La causa de que la revisión de código sea tan eficiente, es porque cuando haces revisiones, ves los problemas no los síntomas. Es decir, mientras revisas el código, piensas sobre lo que el programa debe hacer. Así cuando algo no lo ves correcto, puedes ver el posible problema y rápidamente verificar el código. Puesto que el tiempo transcurrido desde que se detecta el síntoma hasta que se llega al problema, es la mayor parte del coste de encontrar y corregir los defectos durante la compilación y pruebas, las revisiones pueden ahorrar mucho tiempo.

Las revisiones también tienen desventajas. Las dos desventajas principales son que las revisiones de código consumen tiempo y es difícil hacerlas correctamente. La revisión, sin embargo, es una habilidad que se puede enseñar y mejorar con la práctica. Con la experiencia, sin embargo, probablemente encontrarás solamente una media del 75% al 80% de los defectos en un programa. Necesitarás al menos 30 minutos para revisar minuciosamente cada 100 LOC del código fuente. Cuando hagas revisiones mucho más rápido, normalmente encontrarás menos defectos. El resto de este capítulo describe el proceso de revisión de código. Los si-

gientes capítulos discuten las medidas y las técnicas que te ayudarán a mejorar la forma de hacer las revisiones de código y te proporcionan datos para demostrar su efectividad.

13.5

¿POR QUÉ HAY QUE ENCONTRAR PRONTO LOS DEFECTOS?

Hay muchas razones para revisar los programas antes de compilarlos y probarlos. La razón más importante es que no puedes parchear un programa defectuoso y transformarlo posteriormente en un producto de calidad. Una vez que has hecho un programa defectuoso, siempre será defectuoso. Puedes corregir todos los problemas que encuentres y puedes hacer que trabaje de acuerdo con las especificaciones con que lo has probado, pero seguirá siendo un programa defectuoso con muchos parches.

Como ejemplo, supón que estás comprando un coche nuevo. Antes de cerrar el trato, visitas dos fábricas de ensamblaje. En una planta, ves muchos coches bonitos que salen de la línea de producción y van a la de pruebas. Aunque salen grandes coches de la línea de producción, en la fase de pruebas se encuentran una media de 10 defectos por coche. Estos defectos son corregidos y los coches se envían a los comercios.

En la segunda planta, los coches que salen de la línea de producción se parecen a los de la primera. Aquí, sin embargo, en las pruebas solamente se encuentra un defecto por cada 10 coches. Aunque los coches de la segunda planta cuestan un poco más, probablemente los preferirás, sin tener en cuenta cualquier otra diferencia. Sabes que en las pruebas no encontrarás todos los problemas y que si en el proceso de producción del coche ha habido muchos fallos, ese coche probablemente siempre tendrá muchos fallos, independientemente de la cantidad de pruebas finales e inspecciones.

Los programas no son diferentes. Cuando los ingenieros toleran un trabajo defectuoso, harán productos de poca calidad. Una actitud de “estoy tan ocupado, podemos arreglarlo más tarde”, es improbable que produzca un producto de gran calidad. Para hacer software de calidad, cada paso del proceso de desarrollo de software debe ser de gran calidad. Aunque las prácticas rigurosas de calidad pueden parecer costosas, te ahorrarán mucho tiempo.

13.6

EL COSTE DE ENCONTRAR Y CORREGIR DEFECTOS

En los típicos proyectos de software, el producto es dividido en muchos programas elementales o módulos pequeños. Cada ingeniero, desarrolla

uno o más de estos módulos. Después de diseñar el módulo, implementarlo y compilarlo, los ingenieros hacen una prueba inicial o prueba de unidad. Después de estas pruebas de unidad privadas, se combinan los módulos en un gran componente y se hacen pruebas de integración. Se realizan varios niveles de pruebas de componentes antes de que se combinen los componentes en productos para hacer las pruebas del producto. Finalmente, se ensamblan los productos en los sistemas para hacer las pruebas del sistema. Aunque el tipo, duración y complejidad de las pruebas de integración, de componentes, de producto y del sistema variará con el tamaño y complejidad del sistema, se utiliza el mismo proceso general para casi todos los productos software a gran escala.

El coste medio de encontrar y corregir un defecto crece unas 10 veces en cada paso del proceso de desarrollo. Aunque el tiempo de corregir los defectos varía enormemente, estos valores medios muestran, a pesar de todo, los tipos de defectos. Algunos defectos triviales de sintaxis, como un punto y coma mal colocado o errores tipográficos en los nombres pueden pasar la fase de compilación, siendo muy difícil encontrarlos en la fase de pruebas. En la revisión de código encontrarás y corregirás los defectos en una media de 1 a 2 minutos. En las pruebas de unidad iniciales, sin embargo, los tiempos para corregir los defectos tendrán un valor medio de entre 10 y 20 minutos o más. Estos datos corresponden, en su mayor parte, a correcciones que necesitan entre 1 y 2 minutos, y existen unas pocas que necesiten varios minutos o varias horas.

El tiempo de encontrar los defectos en las pruebas de integración, de componentes o del sistema, también variará con el tamaño y la complejidad del sistema. Muchas veces se requiere encontrar y corregir defectos en sistemas grandes y muy complejos. En las pruebas de integración, por ejemplo, cada defecto puede costar una hora o más, y en las pruebas del sistema cada defecto puede costar entre 10 a 40 o más horas de ingeniero. Una vez que los productos son entregados a los clientes, el coste de encontrar y corregir los defectos puede ser mucho mayor, dependiendo de la clase de productos y de los tipos y número de clientes. Mis datos personales de los tiempos de encontrar y corregir los defectos en C++ se muestran en la Figura 13.1. El siguiente ejemplo muestra el coste de esperar hasta que las pruebas eliminén todos los defectos del programa.

- Una empresa pequeña de software comercial desarrolló un programa con varios componentes. Las pruebas de integración realizadas por los ingenieros que estaban entrenados en el PSP duraron un par de semanas. Un componente, sin embargo, se desarrolló por un grupo que no había recibido formación en el PSP y las pruebas de integración se realizaron en varias semanas. El tiempo de las pruebas para encontrar y corregir los defectos fue de 300 horas. Puesto que las pruebas necesitaron mucho más tiempo que el planificado, la entrega al cliente se hizo dos meses más tarde.

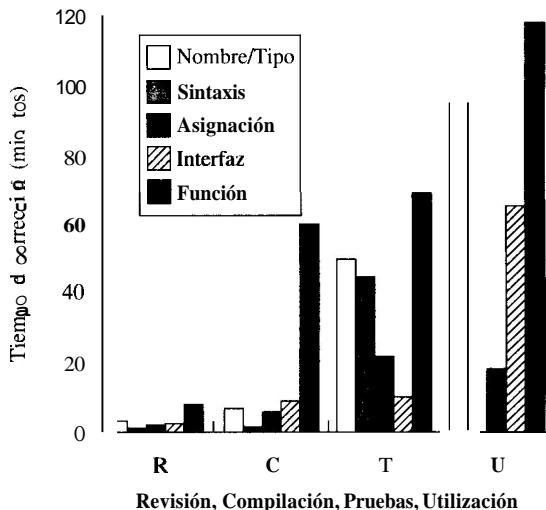


Figura 13.1 Tiempos de corrección de defectos.

- El desarrollo de un sistema aeroespacial, necesitó una media de 40 horas de ingeniero para encontrar y corregir cada defecto en las pruebas del sistema de un sistema de navegación aérea.
- En Digital Equipment Corporation, para un sistema, el tiempo mínimo para encontrar y corregir cada defecto informado por el cliente fue de **88** horas de ingeniero.

Además del coste, una razón de igual importancia para encontrar los defectos al-principio, es que la compilación, depuración y las pruebas tienen una efectividad reducida. Los compiladores son las herramientas más rápidas que tenemos para detectar defectos, pero solamente encuentran alrededor del 90% de los defectos de sintaxis y muy pocos defectos lógicos. La prueba de unidad es normalmente la prueba más efectiva, pero encuentra la mitad de los defectos del programa. Despues de la prueba de unidad, la efectividad de las pruebas disminuye, con las pruebas del sistema, normalmente se encuentran entre un 30% y un 40% de los defectos del producto.

Así, si quieres producir un producto de alta calidad, tendrás que producir un programa sin defectos al principio o esperar dedicarle mucho tiempo en las pruebas.

13.7

EL USO DE LAS REVISIONES PARA ENCONTRAR DEFECTOS

Puedes encontrar difícil creer que controlando y comprobando tus defectos, mejorarás tu trabajo, pero otros estudiantes han reducido de 5 a 10 ve-

ces el número de defectos que encuentran en la compilación y en las pruebas. Lo consiguen, siguiendo los pasos esbozados en este y en los siguientes capítulos. El trabajo de revisar el código es tan adecuado que después de utilizarlo durante este curso y ver sus buenos resultados, probablemente harás de las revisiones una parte normal de tu proceso personal.

El primer paso para hacer las revisiones, es entender la clase de defectos que puedes introducir. Esta es la razón principal de reunir datos de defectos. El tipo de defectos en tu próximo programa, serán muy parecidos a los encontrados en programas anteriores. Esto será cierto mientras continúes desarrollando software de la misma forma. Por otra parte, conforme vayas adquiriendo **más** habilidades y experiencia, o si cambias de proceso, el número y tipo de defectos cambiará.

Puesto que las clases de defectos que introduces podrían ser diferentes a las introducidas por otra persona, **tu** estrategia de revisión debería basarse en tu perfil de defectos personales. Al igual que un mecanógrafo habiloso comete pocos errores de mecanografía, un ingeniero del software experimentado cometerá pocos errores de programación. Mejorarás de forma natural conforme tengas más práctica, pero hay algún punto a partir del cual la mejora es más difícil. Entonces, debes estudiar los defectos. Esto te ayudará a saber cómo mejorar la forma de encontrarlos y corregirlos.

Los objetivos de la revisión de código son encontrar el mayor número de defectos lo más pronto posible en el proceso software. Tú también quieres encontrar cada defecto en el menor tiempo posible. Un guión para hacer una revisión de código se muestra en la Tabla 13.1. Aunque, este resumen es autoexplicativo, es importante que cuando revises el código tengas en cuenta lo siguiente:

- Hacer la revisión antes de la primera compilación.
- Hacer la revisión en un listado impreso del código fuente.
- Registrar cada defecto encontrado en el Cuaderno de Registro de Defectos.
- Durante la revisión, comprueba los tipos de defectos que hayas previamente encontrado en compilaciones y en pruebas. Un método ordenado para hacer esto se describe en el Capítulo 14.

13.8

REVISAR ANTES DE COMPILAR

Hay varias razones para revisar los programas antes de compilarlos. En esencia son:

1. Dedicarás casi el mismo tiempo a hacer una revisión completa de código si la haces antes **o** después de la compilación.

Tabla 13.2 Guión para la revisión de código.

	Criterios de entrada	Comprueba que tienes a mano: <ul style="list-style-type: none"> • Especificación de requisitos. • El diseño del programa. • El código fuente del programa. • Estándares de codificación.
1	Procedimiento de revisión	Primero escribe el código fuente del programa completo. Antes de compilar o probar el programa, imprime un listado del código fuente. A continuación, haz la revisión del código. Durante la revisión del código, chequea cuidadosamente cada línea del código fuente para encontrar y corregir tantos defectos como puedas.
2	Corregir los defectos	Corregir todos los defectos encontrados. Comprobar las correcciones para asegurar que son correctas. Anotar los defectos en el Cuaderno de Registro de Defectos.
3	Revisar el ámbito	Verifica que el diseño del programa satisface todas las funciones descritas en la especificación. Verifica que el código fuente implementa todo el diseño.
4	Revisar la lógica del programa	Verifica que el diseño lógico es correcto. Verifica que el programa implementa correctamente el diseño lógico.
5	Comprobar los nombres y los tipos	Verifica que todos los nombres y tipos son correctamente declarados y utilizados. Chequea la correcta declaración de los tipos de dato integer, long integer y float.
6	Comprobar todas las variables	Asegúrate que cada variable está inicializada. Chequea los problemas de desbordamiento, underflow y fuera de rango.
7	Comprobar la sintaxis del programa	Verifica que el código fuente cumple todas las especificaciones del lenguaje.
	Criterios de salida	Al finalizar debes tener: <ul style="list-style-type: none"> • El código fuente terminado y corregido. • El Cuaderno de Registro de Tiempos completo. • El Cuaderno de Registro de Defectos completo.

2. Revisar primero te ahorrará mucho tiempo de compilación. Antes de hacer una revisión de código, los ingenieros normalmente dedican entre el 12% y el 15% de su tiempo de desarrollo a la compilación. Una vez que aprenden a hacer la revisión de código, sus tiempos de compilación caen hasta un **3%** o menos.
3. Una vez que los ingenieros han compilado sus programas, normalmente sus revisiones no son tan completas.

4. La compilación es igualmente efectiva antes o después de la revisión de código.
5. La experiencia indica que cuando los programas tienen muchos defectos durante la compilación, generalmente tienen muchos defectos en las pruebas.

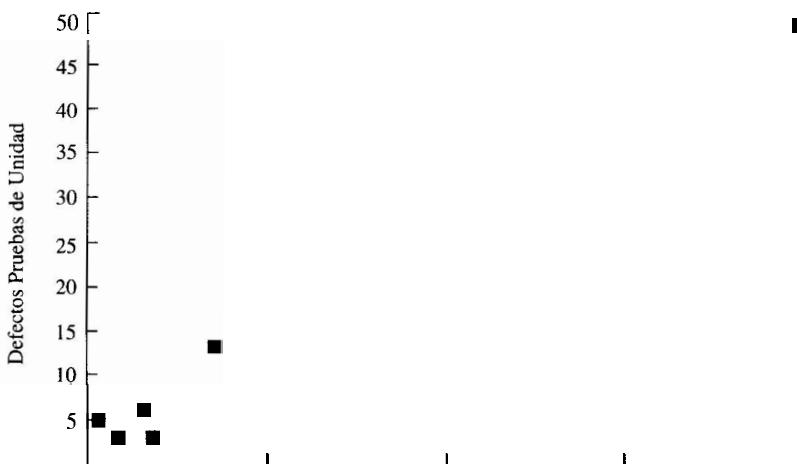
El argumento para revisar antes de compilar es sencillo: “hazlo de esta forma y lo verás”. Si, aún después de este curso, quisieras intentar compilar antes de hacer la revisión, entonces escribe unos pocos programas de cada forma y compara su rendimiento. Entonces, utiliza los métodos descritos en los Capítulos 18 y 19, evalúa los datos del proceso y del producto para ver qué aproximación es más efectiva. Un punto importante es no seguir tu intuición o hacer lo que todo el mundo hace. Reúne los datos de tu propio trabajo y toma una decisión lógica basada en estos hechos.

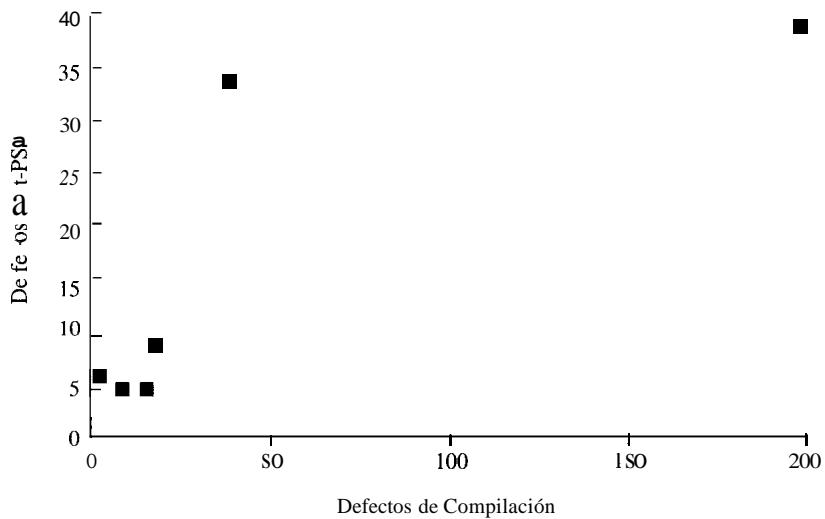
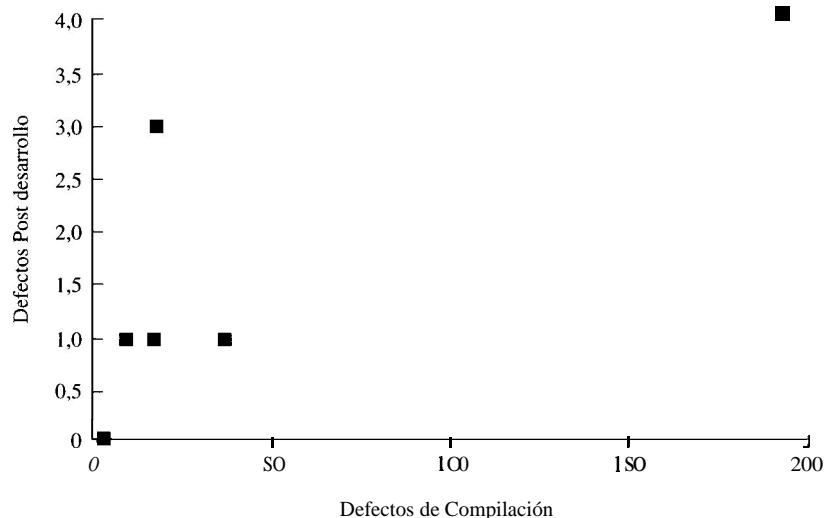
13.9

DATOS SOBRE DEFECTOS DE COMPILACIÓN Y PRUEBAS

Los ingenieros de un proyecto habían sido entrenados en el PSP. Ellos desarrollaron seis componentes de un nuevo producto y cada componente tenía de 600 a **2500 LOC**. En su proceso hicieron pruebas de unidad después de codificar y compilar, y después una prueba de desarrollo. El producto se entregó al cliente y este informó sobre los defectos, que fueron registrados como defectos de postdesarrollo.

Los datos de los defectos de compilación y pruebas para estos componentes se muestran en las Figuras 13.2, 13.3 y 13.4. Un ingeniero que



**Figura 13.3** Defectos de compilación vs. post-PSP.**Figura 13.4** Defectos de compilación vs. postdesarrollo.

se había opuesto firmemente a hacer una revisión de código, encontró cerca de 200 defectos durante la compilación. Como se muestra en la Figura 13.2, sus pruebas tenían muchos más defectos que la de los otros componentes. Lo que es interesante, es que, este mismo componente también tenía más defectos en las pruebas de integración y en las del sistema. Aunque el cliente había hecho solamente unas breves pruebas de aceptación del total de componentes, el mismo componente volvía a tener más defectos. Finalmente, este componente defectuoso necesitó el doble de

tiempo de lo planificado para desarrollarlo. Los otros componentes fueron entregados en el momento planificado o antes, mientras que aquel componente se entregó cinco semanas después.

El trabajo cuidadoso se amortiza. Cuando los ingenieros sienten personalmente la responsabilidad de la calidad de sus programas, ellos no dependerán de los compiladores o de otras herramientas para encontrar sus defectos. Cuando te comprometes a hacer un producto de calidad, tu compromiso aparecerá en el número de defectos que encuentras en la compilación, en las pruebas y en la calidad de tus programas acabados.

13.10 ACTUALIZACIÓN DE LA TABLA RESUMEN DEL PLAN DEL PROYECTO DEL PSP

El proceso del PSP se mejora ahora con la inclusión de la revisión de código. Esto se hace en el guión del proceso modificado que se muestra en la Tabla 13.2. La tabla del Resumen del Plan del Proyecto se actualiza también, tal como se muestra en las Tablas 13.3, 13.4 y en el ejemplo resumen de la Tabla 13.5.

Estas tablas son autoexplicativas, pero hay un punto que merece mención especial. Al completar las columnas de Hasta la Fecha y % Hasta la Fecha para este proceso, recuerda que la introducción de la revisión del código cambiará tus distribuciones de tiempo y de los defectos. Cuando planifiques un nuevo programa, puedes querer ajustar el tiempo calculado estimado para contabilizar estos cambios.

13.11 OTRAS CLASES DE REVISIONES

En las organizaciones de software, una práctica común es tener a varios ingenieros revisando programas unos a otros. Esto se llama revisiones en pareja o inspecciones. Las inspecciones bien hechas normalmente encuentran del 50 al 70% de los defectos de un programa. Aunque las inspecciones suponen mucho tiempo, pueden ser muy efectivas para encontrar defectos. La razón es que los ingenieros, a menudo, tienen dificultad en ver sus propios errores de diseño. Ellos crean el diseño y saben qué es lo que pretenden hacer. Si su idea era defectuosa o hacen un diseño erróneo o una suposición en la implementación, a menudo, tienen dificultad en detectarlo. Las inspecciones pueden ayudar a superar este problema. Los datos de los tiempos para encontrar defectos en las inspecciones se muestran en la Tabla 13.6.

Tabla 13.2 Guión del proceso del PSP.

	Propósito	Guiarte en el desarrollo de pequeños programas.
	Criterios de entrada	La descripción del problema. Tabla Resumen del Plan del Proyecto del PSP. Datos de tamaños y tiempos reales para pequeños programas. Cuaderno de Registro de Tiempos. Cuaderno de Registro de Defectos.
1	Planificación	Obtén una descripción de las funciones del programa. Estima las LOC Máx., Mín., total requeridas. Determina los Minutos/LOC. Calcula los tiempos de desarrollo Máx., Mín., y total. Anota los datos del plan en la tabla Resumen del Plan del Proyecto. Anota el tiempo de planificación en el Cuaderno de Registro de Tiempos.
2	Diseño	Diseña el programa. Anota el diseño en el formato especificado. Anota el tiempo de diseño en el Cuaderno de Registro de Tiempos.
3	Codificación	Implementa el diseño. Utiliza un formato estándar para introducir el código. Anota el tiempo de codificación en el Cuaderno de Registro de Tiempos.
4	Revisión de código	<i>Revisar completamente el código fuente.</i> <i>Seguir el guión de revisión de código.</i> <i>Corregir y registrar todos los defectos encontrados.</i> <i>Registrar el tiempo de revisión en el Cuaderno de Registro de Tiempos.</i>
5	Compilación	Compila el programa. Corrige y registra todos los errores encontrados. Anota el tiempo de compilación en el Cuaderno de Registro de Tiempos.
6	Pruebas	Prueba el programa. Corrige y anota todos los errores encontrados. Anota el tiempo de pruebas en el Cuaderno de Registro de Tiempos.
7	Postmortem	Completa la tabla de Resumen del Plan del Proyecto con los datos de tiempo, tamaño y defectos reales. Anota el tiempo postmortem en el Cuaderno de Registro de Tiempos.
	Criterios de salida	Programa probado a fondo. Diseño adecuadamente documentado. Listado completo del programa. Resumen del Plan del Proyecto completo. Cuadernos de tiempos y defectos completos.

Tabla 13.3 Resumen del plan del proyecto del PSP.

Estudiante _____	Fecha _____		
Programa _____	Programa # _____		
Profesor _____	Lenguaje _____		
<i>Resumen</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la Fecha</i>
Minutos/LOC	_____	_____	_____
LOC/Hora	_____	_____	_____
Defectos/KLOC	_____	_____	_____
Rendimiento	_____	_____	_____
V/F	_____	_____	_____
Tamaño Programa (LOC):			
Total Nuevo & Cambiado	_____	_____	_____
Tamaño Máximo	_____	_____	_____
Tamaño Mínimo	_____	_____	_____
<i>Tiempo por Fase (min.)</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la Fecha</i>
Planificación	_____	_____	_____
Diseño	_____	_____	_____
Codificación	_____	_____	_____
Revisión del código	_____	_____	_____
Compilación	_____	_____	_____
Pruebas	_____	_____	_____
Postmorten	_____	_____	_____
Total	_____	_____	_____
Tiempo Máximo	_____	_____	_____
Tiempo Mínimo	_____	_____	_____
<i>Defectos introducidos</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la Fecha</i>
Planificación	_____	_____	_____
Diseño	_____	_____	_____
Codificación	_____	_____	_____
Revisión del código	_____	_____	_____
Compilación	_____	_____	_____
Pruebas	_____	_____	_____
Total	_____	_____	_____
<i>Defectos eliminados</i>	<i>Plan</i>	<i>Real</i>	<i>Hasta la Fecha</i>
Planificación	_____	_____	_____
Diseño	_____	_____	_____
Codificación	_____	_____	_____
Revisión del código	_____	_____	_____
Compilación	_____	_____	_____
Pruebas	_____	_____	_____
Total	_____	_____	_____
			<i>Def./Hora</i>

Para ejercicios muy pequeños de clase, las inspecciones generalmente no están justificadas, pero para grandes proyectos o cualquier programa industrial, las inspecciones deberían hacerse siempre. La estrategia Óptima, es hacer una revisión de código personal antes de compilar, y compilar después el programa. A continuación, antes de cualquier prueba, haz una inspección. Aunque este libro no discute posteriormente las inspecciones, hay varias referencias útiles sobre este asunto [Fagan, Gilb, Humphrey 89].

Tabla 13.4 Instrucciones del resumen del plan del proyecto del PSP.

Propósito	Esta tabla trata los datos estimados y reales de los proyectos de una forma cómoda y fácilmente recuperable.
Cabecera	<p>Introduce los siguientes datos:</p> <ul style="list-style-type: none"> • Tu nombre y fecha de hoy. • Nombre y número de programa. • Nombre del profesor. • El lenguaje que utilizarás para escribir el programa.
Minutos/LOC	<p>Antes de iniciar el desarrollo:</p> <ul style="list-style-type: none"> • Escribe los Minutos/LOC planificados para este proyecto. Utiliza la velocidad Hasta la Fecha de un programa reciente del Cuaderno de Trabajos o del Resumen del Plan del Proyecto. <p>Después del desarrollo:</p> <ul style="list-style-type: none"> • Divide el tiempo total de desarrollo por el tamaño real del programa para obtener los Minutos/LOC reales y los Minutos/LOC Hasta la Fecha. • Por ejemplo, si el proyecto se hizo en 196 minutos e hiciste 29 LOC, los Minutos/LOC serían $196/29 = 6,76$.
LOC/Hora	<p>Antes de iniciar el desarrollo:</p> <ul style="list-style-type: none"> • Calcula las LOC por hora planificada para este programa dividiendo 60 por los Minutos/LOC de la casilla de Plan. <p>Después del desarrollo:</p> <ul style="list-style-type: none"> • Para LOC/Hora Real y Hasta la Fecha divide 60 por Minutos/LOC Reales y Hasta la Fecha. • Para los 6,76 Minutos/LOC Reales, tenemos $60/6,76 = 8,88$ LOC/Hora Reales.
Tamaño Programa (LOC)	<p>Antes de iniciar el desarrollo:</p> <ul style="list-style-type: none"> • Escribe bajo la columna plan, el valor estimado de Total Nuevas & Cambiadas, Máximo y Mínimo. <p>Después del desarrollo:</p> <ul style="list-style-type: none"> • Cuenta y anota las LOC Nuevas & Cambiadas Reales. • Para la columna Hasta la Fecha, añade LOC reales Nuevas & Cambiadas a las LOC Hasta la Fecha Nuevas & Cambiadas de programas anteriores.
Tiempo por fase Plan	<p>Para el tiempo total de desarrollo, multiplica el valor de las LOC Total Nuevas & Cambiadas por Minutos/LOC.</p> <p>Para el tiempo Máximo, multiplica el tamaño Máximo por los Minutos/LOC.</p> <p>Para el tiempo Mínimo, multiplica el tamaño Mínimo por los Minutos/LOC.</p> <p>Del Resumen del Plan del Proyecto de un programa reciente, busca los valores de % Hasta la Fecha para cada fase.</p> <p>Utilizando el % Hasta la Fecha de programas anteriores calcula el tiempo planificado para cada fase.</p>
Real	<p>Una vez acabado el trabajo, anota el tiempo real en minutos que has gastado en cada fase del desarrollo.</p> <p>Obtén estos datos del Cuaderno de Registro de Tiempos.</p>

(Continúa)

Tabla 13.4 Instrucciones del resumen del plan del proyecto del PSP. (Continuación)

Hasta la Fecha	Para cada fase, anota la suma del tiempo real y el tiempo Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, multiplica por 100 el tiempo Hasta la Fecha y lo divides por el total del tiempo Hasta la Fecha.
Defectos Introducidos Reales	Después del desarrollo, localiza y anota el número real de defectos introducidos en cada fase.
Hasta la Fecha	Para cada fase, anota la suma de los defectos reales y los Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, multiplica por 100 los Defectos Hasta la Fecha para esa fase y divídelos por el total de defectos Hasta la Fecha.
Defectos Eliminados Reales	Después del desarrollo, localiza y anota el número real de defectos eliminados en cada fase.
Hasta la Fecha	Para cada fase, escribe la suma de los defectos reales y los defectos Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, multiplica por 100 los Defectos Hasta la Fecha para esa fase y divídelos por el total de defectos Hasta la Fecha.

Tabla 13.6 Horas para encontrar un defecto.

Referencia	Inspección	Prueba	Utilización
Ackerman	1	2-10	
O'Neill	0,26		
Ragland		20	
Russell	1	2-4	33
Shooman	0,6	3,05	
VanGenuchten	0,25	8	
Weller	0,7	6	

RESUMEN

Este capítulo describe cómo y porqué encontrar defectos al principio del desarrollo. El factor más importante en la calidad de un programa es el compromiso personal del ingeniero con la calidad. Con este compromiso, los ingenieros eliminan los defectos al principio, antes de compilar o probar sus programas.

Tabla 11.5 Resumen del plan del proyecto del PSP.

Estudiante	Estudiante X		Fecha	4/12/96
Programa			Programa#	11
Profesor	Sr. Z		Lenguaje	Ada
Resumen	Plan	Real	Hasta la fecha	
Minutos/LOC	6,50	5,88	6,30	
LOC/Hora	9,23	10,20	9,52	
Defectos/KLOC				
Rendimiento				
V/F				
Tamaño Programa(LOC):				
Total Nuevo & Cambiado	53	40	153	
Tamaño Máximo	70			
Tamaño Mínimo	35			
Tiempo por Fase (min.)	Plan	Real	Hasta la Fecha	% Hasta la Fecha
Planificación	17	15	40	5,0
Diseño	20	20	03	8,6
Codificación	156	132	440	45,6
Revisión del código				
Compilación	0	36	36	3,7
Pruebas	35	7	77	8,0
Postmorten	03	30	203	21,1
Total	345	202	964	100,0
Tiempo Máximo	468			
Tiempo Mínimo	220			
Defectos introducidos	Plan	Real	Hasta la Fecha	% Hasta la Fecha
Planificación				
Diseño	1	3	21,4	
Codificación	5	11	78,6	
Revisión del código				
Compilación				
Pruebas				
Total	6	14	100	
Defectos eliminados	Plan	Real	Hasta la Fecha	% Hasta la Fecha
Planificación				
Diseño				
Codificación				
Revisión del código				
Compilación	3	3	21,4	
Pruebas	2	8	57,2	
Total	1	3	21,4	
	6	14	100,0	

El principal método para eliminar defectos introducidos con el PSP es la revisión de código personal. Aquí, primero imprimes un listado del programa y lo revisas línea por línea para arreglar tantos defectos como puedas encontrar. Para ser más eficiente, mira aquellos tipos de defectos que te hayan aparecido en tus errores de compilación y en las pruebas de programas anteriores. Puesto que las personas tienden a repetir los mismos errores, tus datos de los defectos te ayudarán a encontrar todos o muchos de los defectos de los programas, antes de que los compiles y les hagas pruebas.

La experiencia ha mostrado que cuando los ingenieros revisan a fondo sus códigos antes de la primera compilación, reducen el tiempo de compilación en un 10% del tiempo de desarrollo y el tiempo de pruebas, aún más. Aunque una revisión a fondo del código lleva su tiempo, dichas revisiones reducirán el tiempo total al menos, tanto el tiempo como el empleado en las revisiones y se producirán productos de mejor calidad.

EJERCICIO 13

Para el próximo programa, revisa el código fuente antes de compilarlo y probarlo. Sigue el guión de revisión de código mostrado en la Tabla 13.1. Registra todos los defectos encontrados en el Cuaderno de Registro de Defectos y anota los datos del proyecto en la tabla del Resumen del Plan del Proyecto.

Entrega copias del Cuaderno de Registro de Tiempos y de las hojas del Resumen Semanal de Actividades que no hayas entregado previamente. También, entrega copias completas del Resumen del Plan del Proyecto y del Cuaderno de Registro de Defectos para cada programa que desarrolles. Incluye los tiempos de desarrollo planificados y reales y los defectos reales introducidos y eliminados.

REFERENCIAS

[Ackerman] Ackerman, Frank A., Lynne S. Buchwald, and Frank H. Lewski. "Software Inspections: An Effective Verification Process". *IEEE Software*, May 1989, pp. 31-36.

[Fagan] Fagan, Michael. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal*, vol. 15, no. 3, 1976.

[Gilb 93] Gilb, Tom, and Dorothy Graham. *Software Inspection*. Edited by Susannah Finzi, Reading, MA: Addison-Wesley, 1993.

[Humphrey 89] Humphrey, W.S. *Managing the Software Process*. Reading, MA: Addison-Wesley, 1989.

[O'Neill] O'Neill, Don. Comunicación personal.

[Ragland] Ragland, Bryce. "Inspections Are Needed Now More Than Ever". *Journal of Defense Software Engineering*, 38. Software Technology Support Center, U.S. Department of Defense, Nov. 1992.

[Russell] Russell, Gien W. "Experience with Inspections in Ultralarge-Scale Development." *IEEE Software*, Jan. 1991, pp. 25-31.

[Shooman] Shooman, M. L., and M. I. Bolsky. "Types, Distribution, and Test and Correction Times for Programming Errors." *Proceedings of the 1975 Conference on Reliable Software*. Catalog no. 75 CHO 940-7CST, IEEE, New York: p. 347.

[vanGenuchten] vanGenuchten, Michael. Comunicación personal.

[Weller] Weller, E. F., "Lessons Learned from Two Years of Inspection Data." *IEEE Software*, Sept. 1993, pp. 38-45.

CAPITULO 14

Listas de comprobación para la revisión de código

La clave para realizar una revisión de código efectiva es tener un procedimiento de revisión eficiente. Este capítulo describe las listas de comprobación para la revisión de código, y explica cómo pueden ayudarte, para que de una forma rápida y eficiente, encuentres los defectos en tus programas y hagas una lista de comprobación para tu uso personal. Como ejercicio, diseñarás una lista de comprobación para los defectos que normalmente introduzcas y la utilizarás en la revisión de tus programas.

14.1

¿POR QUÉ AYUDAN LAS LISTAS DE COMPROBACIÓN?

Una lista de comprobación contiene una serie de pasos de procedimiento que quieras seguir de forma precisa. Cuando las personas tienen cosas importantes que quieren hacer exactamente tal y como están especificadas, a menudo, utilizan las listas de comprobación. Los pilotos de líneas aéreas, por ejemplo, las utilizan para hacer una comprobación pre-vuelo antes de despegar. Aunque hayan hecho una comprobación del mismo avión una hora antes, la vuelven a hacer. Un estudio de **los** accidentes en una base de las Fuerzas Aéreas de los EE.UU., encontró que en cada caso, la lista de comprobación pre-vuelo no se había seguido rigurosamente. Otro ejemplo de una lista de comprobación completa y compleja es la cuenta

atrás utilizada por la **NASA** antes de cada lanzamiento espacial. Este procedimiento se realiza durante varios días y sigue cientos de pasos. Es tan complejo, que se utilizan computadoras para controlar el progreso de la cuenta atrás.

Cuando es esencial encontrar y corregir cada defecto en un programa, debes seguir un procedimiento preciso. Una lista de comprobación te puede ayudar a asegurarte de que se sigue el procedimiento. En este capítulo, trataremos una clase muy especial de lista de comprobación: una diseñada para ayudarte a encontrar los defectos cuando hagas una revisión de código de un programa que has escrito. Verás cómo construyes una lista de comprobación para la revisión de código, que se adapta para encontrar los defectos que te han causado anteriormente muchos problemas.

Las listas de comprobación también pueden ser una fuente de ideas. Cuando sigues una lista de comprobación personal, sabes cómo revisar tu código. Si utilizas la lista correctamente, también sabes cuantos defectos encuentras en cada paso de dicha lista. Comparar tu lista de comprobación con las de otros ingenieros, te puede sugerir aproximaciones útiles para la revisión.

La lista de comprobación encapsula la experiencia personal. Utilizándola con regularidad y mejorándola, mejorarás en la detección de los defectos de tus programas. La lista de comprobación también te ayudará a encontrar estos defectos en menos tiempo.

14.2

UN EJEMPLO DE LISTA DE COMPROBACIÓN PARA LA REVISIÓN DE CÓDIGO

La lista de comprobación para la revisión de código que diseñé para revisar mis programas en C++ se muestra en la Tabla 14.1. Una lista de comprobación similar para el lenguaje Ada se muestra en la Tabla 14.2. Estas listas de comprobación sugieren un número de puntos a considerar, conforme desarrolles y utilices tu propia lista de comprobación personal.

Un primer paso muy útil es asegurar que el código implementa todas las funciones incluidas en el diseño. En grandes programas, es fácil descuidar la codificación de algún procedimiento u operación. Dichos descuidos son errores comunes y pueden, ocasionalmente, pasar las siguientes etapas de revisión, compilación y pruebas. Los descuidos generalmente son fáciles de encontrar con una lista de comprobación.

Comprobaciones completas para `includes` (o `withs`), inicialización, llamadas a procedimientos y nombres, son también efectivas. Estas, son las áreas de problemas comunes que deberías comprobar a no ser que los datos históricos te indicasen que tú NUNCA has cometido dichos errores.

Tabla 14.1 Lista de comprobación y guía para la revisión de código en C++**Nombre del Programa y #:**

Propósito	Guía para realizar una revisión de código efectiva	#	#	#	#	Hasta la Fecha	% Hasta la Fecha
Método	Cuando completes cada paso de la revisión, anota el número de defectos que has encontrado de cada tipo en la casilla de la derecha. Si no hay ninguno, anota un control en la casilla de la derecha. Completa la lista de comprobación para un programa, clase, objeto, o método antes de comenzar a revisar la siguiente.						
Completo	Verifica que todas las funciones del diseño están programadas.						
Includes	Verifica que los <code>includes</code> están completos						
Inicialización	Comprobar la inicialización de parámetros y variables: <ul style="list-style-type: none"> * Al inicio del programa. * Al comenzar cada bucle. * En la entrada a un procedimiento o función. 						
Llamadas	Comprobar los formatos de las llamadas a las funciones: <ul style="list-style-type: none"> ▪ Punteros. ▪ Parámetros. ▪ Utilización de '&' 						
Nombres	Comprobar la ortografía de los nombres y su utilización: <ul style="list-style-type: none"> ▪ ¿Es consistente? * ¿Está dentro del ámbito declarado? • ¿Todas las estructuras/clases utilizan la referencia '.'? 						
Cadenas de caracteres	Comprobar que todas las cadenas de caracteres: <ul style="list-style-type: none"> ▪ Están identificadas por punteros. ▪ Terminan en NULL. 						
Punteros	Comprobar los punteros: <ul style="list-style-type: none"> * Están inicializados a NULL. * Solamente se eliminan después de new. ▪ Siempre se eliminan después de utilizarlos. 						
Formato de salida	Comprobar el formato de salida: <ul style="list-style-type: none"> ▪ ¿Es adecuado el salto de línea? * ¿Es adecuado el espaciado? 						
() Parejas	Asegurarse que los corchetes () son los adecuados y están balanceados.						
Operadores lógicos	Verificar la utilización correcta de ==, !=, , etc. Comprobar que cada función lógica tiene () .						
Comprobación línea a línea	Comprobar cada línea de código: <ul style="list-style-type: none"> ▪ Sintaxis de las instrucciones. ▪ Signos de puntuación adecuados. 						
Estándares	Asegurar que cada programa se adapta a los estándares de codificación.						
Apertura y cierre de ficheros	Verifica que todos los ficheros son: <ul style="list-style-type: none"> ◦ Declarados de forma adecuada. ▪ Abiertos. ▪ Cerrados. 						
Global	Hacer una revisión global al programa para comprobar los resultados del sistema y problemas inesperados.						
Totales							

Tabla 14.2 Lista de comprobación y guía para la revisión de código en Ada.

Nombre del Programa y #:

Propósito	Guía para realizar una revisión de código efectiva	#	#	#	lasta la fecha	% Hasta la Fecha
Método	Cuando completes cada paso de la revisión, anota el número de defectos que has encontrado de cada tipo en la casilla de la derecha. Si no hay ninguno, anota un control en la casilla de la derecha. Completa la lista de comprobación para un programa, clase, objeto, o método antes de comenzar a revisar la siguiente.					
Completo	Verifica que todas las funciones del diseño están programadas.					
Includes	Verifica que las sentencias <code>with</code> están completadas					
Inicialización	Comprobar la inicialización de parámetros y variables: <ul style="list-style-type: none">▪ Al inicio del programa.• Al comenzar cada bucle.▪ En la entrada a un procedimiento o función.					
Llamadas	Comprobar los formatos de las llamadas a los procedimientos: <ul style="list-style-type: none">▪ Signos de puntuación.▪ Parámetros.					
Nombres	Comprobar la ortografía de los nombres y su utilización: <ul style="list-style-type: none">▪ ¿Es consistente?▪ ¿Está dentro del ámbito declarado?▪ ¿Todas las estructuras y paquetes utilizan la referencia '.'?					
Cadenas de caracteres	Comprobar que todas las cadenas de caracteres hacen una utilización adecuada de la partición.					
Punteros	Comprobar los punteros: <ul style="list-style-type: none">• Solamente se eliminan después de new.▪ Siempre se eliminan después de utilizarlos.					
Formato de salida	Comprobar el formato de salida: <ul style="list-style-type: none">• ¿Es adecuado el salto de línea?▪ ¿Es adecuado el espaciado?					
() Parejas	Asegurarse que los corchetes () son los adecuados y están balanceados.					
Operadores lógicos	Verificar la utilización correcta de todas las operaciones lógicas Comprobar que cada función lógica tiene ().					
Comprobación línea a linea	Comprobar cada línea de código: <ul style="list-style-type: none">* Sintaxis de las instrucciones.• Puntuación adecuada.					
Estándares	Asegurar que cada programase adapta a los estándares de codificación.					
Apertura y cierre de ficheros	Verifica que todos los ficheros son: <ul style="list-style-type: none">▪ Declarados de forma adecuada.• Abiertos.• Cerrados.					
Global	Hacer una revisión global al programa para comprobar los resultados del sistema y problemas inesperados.					
Totales						

También considera comprobar el código frente a estándares de codificación, para asegurarte que no has omitido comentarios clave, utilizado un formato inadecuado u omitido información importante del proceso o del producto. Aunque los estándares de codificación pueden parecer que no tienen importancia, pueden ser de gran ayuda para trabajos posteriores, tales como la corrección de un programa, la mejora o la reutilización. Puesto que estas son las actividades principales para muchas organizaciones de software, deberías adquirir el hábito de seguir estándares de codificación (véase la sección 14.7).

El principal peligro con las listas de comprobación es que generalmente encuentras lo que buscas. Aunque esto es útil, el problema es que, si solamente haces las pruebas de la lista de comprobación, solamente encontrarás lo que está en dicha lista. A menudo, sin embargo, los problemas serios aparecen de forma inesperada, tales como interacciones globales entre programas, cuestiones de coordinación imprevistas, problemas complejos de utilización de memoria o condiciones de funcionamiento inusuales. Por ello es una buena idea hacer al menos una revisión general del programa para buscar lo inesperado. Cuando lo hagas, intenta mirarlo desde la perspectiva del sistema o del usuario.

14.3

UTILIZACIÓN DE UNA LISTA DE COMPROBACIÓN PARA LA REVISIÓN DE CÓDIGO

Para utilizar una lista de comprobación para la revisión de código, lee cada apartado sucesivamente y haz las acciones prescritas, tal y como están establecidas. Cuando completes cada acción, márcala en la lista de comprobación. Al final, revisa toda la lista de comprobación para asegurarte que has comprobado cada apartado. Si no lo has hecho, vuelve atrás y haz las acciones omitidas, compruébalas y revisa de nuevo la lista para asegurarte de que no has omitido nada. Al utilizar una lista de comprobación, te podrían ser útiles las siguientes prácticas:

1. Revisa todo el programa para cada apartado de la lista de comprobación. Con la lista de la Tabla 14.1, por ejemplo, revisa en primer lugar todo el programa, para asegurarte de que se ha implementado todo el diseño. Durante esta revisión, si observas otros defectos, corrígelos Tu intención, sin embargo, es comprobar todo el programa frente al diseño. A continuación, revisa el siguiente apartado de la lista de comprobación y así sucesivamente.
2. Cuando encuentres defectos durante cualquier comprobación, anota ese hecho con una marca en la primera columna libre de la derecha #. Para un segundo defecto, se escribirá una segunda marca en la misma columna. Así, después de una revisión completa, pue-

des volver atrás y ver cuántos defectos has encontrado en cada paso de la revisión.

3. Después de completar cada comprobación, si no has encontrado defectos, se escribe una x en la primera casilla no utilizada de la columna # .
4. Cuando revises un programa con varias funciones, objetos o procedimientos, es una buena idea revisar cada entidad por separado. Es decir, revisa el procedimiento y escribe una x o el número de defectos en la primera columna de la derecha #. Para el segundo procedimiento, se hace lo mismo y se anota el resultado de la revisión en la segunda columna #. Continúa así hasta que hayas revisado todas las funciones, objetos o procedimientos.
5. Como se ha observado anteriormente, siempre es una buena idea hacer un examen final global de todo el programa para buscar lo inesperado, nuevas clases de problemas, o problemas del sistema o del usuario.

El proceso general a seguir con una lista de comprobación para la revisión de código se muestra actualizado en el guión para la revisión de código de la Tabla 14.3. En la Tabla 14.4, se muestra actualizado el guión del Proceso del PSP. Estos guiones tienen solamente un par de cambios con respecto al último capítulo, para incluir la lista de comprobación y la exigencia de que se complete cuando hagas la revisión.

14.4

LA ELABORACIÓN DE UNA LISTA DE COMPROBACIÓN PERSONAL

Para hacer una lista de comprobación para la revisión de código, revisa en primer lugar los datos de los defectos y observa cuáles de esos defectos provocan los mayores problemas. Aunque inicialmente tendrás una cantidad limitada de datos de defectos, tendrás más con cada nuevo programa que hagas. Para ser más efectivo, recuerda que la lista de comprobación debe ser diseñada por ti, para el lenguaje que utilices y para los tipos de defectos que normalmente encuentras y omites. Aunque para comenzar cualquier lista te puede ayudar, probablemente no será tan eficiente si no se adapta a tus necesidades específicas.

A continuación, indicamos algunas instrucciones que te pueden ayudar a hacer una lista de comprobación personal útil:

1. Hacer una lista clasificada numéricamente con los defectos en cada fase del proceso software. Véase, por ejemplo, los datos del Estudiante X en la Tabla 14.5. En la parte inferior de la tabla podemos ver los defectos encontrados por fase para cada programa.

Tabla 14.3 Guión para la revisión de código.

Criterios de entrada	Comprueba que tienes a mano: <ul style="list-style-type: none"> • Especificación de requisitos. • El diseño del programa. • El código fuente del programa. • Estándares de codificación. • <i>Una copia de la lista de comprobación para la Revisión de Código.</i>
Método	<i>Utiliza la Lista de Comprobación para la Revisión de Código. Sigue las instrucciones de la lista de comprobación durante la revisión.</i> <i>Una vez completada la revisión, rellena las columnas Hasta la Fecha y % Hasta la Fecha, y la fila de Totales.</i>
1 Procedimiento de revisión	Primero escribe el código fuente del programa completo. Antes de compilar o probar el programa, imprime un listado del código fuente. A continuación, haz la revisión del código. Durante la revisión del código, comprueba cuidadosamente cada línea del código fuente para encontrar y corregir tantos defectos como puedas.
2 Corregir los defectos	Corregir todos los defectos encontrados. Comprobar las correcciones para asegurarte que son correctas. Anotar los defectos en el Cuaderno de Registro de Defectos.
3 Revisar el ámbito	Verifica que el diseño del programa satisface todas las funciones descritas en la especificación. Verifica que el código fuente implementa todo el diseño.
4 Revisar la lógica del programa	Verifica que el diseño lógico es correcto. Verifica que el programa implementa correctamente el diseño lógico.
5 Comprobar los nombres y los tipos	Verifica que todos los nombres y tipos son correctamente declarados y utilizados. Comprueba la correcta declaración de los tipos de dato integer, long integer y float
6 Comprobar todas las variables	Asegúrate que cada variable está inicializada. Comprueba los problemas de desbordamiento, underflow y fuera de rango.
7 Comprobar la sintaxis del programa	Verifica que el código fuente cumple todas las especificaciones del lenguaje.
8 Revisar el programa	Haz una revisión global del programa para comprobar cuestiones del sistema y problemas inesperados.
Criterios de salida	Al finalizar debes tener: <ul style="list-style-type: none"> • El código fuente terminado y corregido. • El Cuaderno de Registro de Tiempos completo. • El Cuaderno de Registro de Defectos completo.

Tabla 14.4 Guión del proceso del PSP.

	Propósito	Guiarte en el desarrollo de pequeños programas.
	Entradas requeridas	<p>La descripción del problema.</p> <p>Tabla Resumen del Plan del Proyecto PSP.</p> <p><i>Una copia de la lista de comprobación para la revisión de código.</i></p> <p>Datos de tamaños y tiempos reales de programas anteriores.</p> <p>Cuaderno de Registro de Tiempos.</p> <p>Cuaderno de Reaistro de Defectos.</p>
1	Planificación	<p>Obtén una descripción de las funciones del programa.</p> <p>Estima las LOC Máx., Mín., total requeridas.</p> <p>Determina los Minutos/LOC</p> <p>Calcula los tiempos de desarrollo Máx., Mín., y total.</p> <p>Escribe los datos del plan en la tabla Resumen del Plan del Proyecto.</p> <p>Anota el tiempo de planificación en el Cuaderno de Registro de Tiempos.</p>
2	Diseño	<p>Diseña el programa.</p> <p>Anota el diseño en el formato especificado.</p> <p>Anota el tiempo de diseño en el Cuaderno de Registro de Tiempos.</p>
3	Codificación	<p>Implementa el diseño.</p> <p>Utiliza un formato estándar para introducir el código.</p> <p>Anota el tiempo de codificación en el Cuaderno de Registro de Tiempos.</p>
4	Revisión de código	<p>Revisar completamente el código fuente.</p> <p>Seguir el guión de revisión de código y de la lista de comprobación.</p> <p>Corregir y registrar todos los defectos encontrados.</p> <p>Registrar el tiempo de revisión en el Cuaderno de Registro de Tiempos.</p>
5	Compilación	<p>Compila el programa.</p> <p>Corrige y anota todos los errores encontrados.</p> <p>Anota el tiempo de compilación en el Cuaderno de Registro de Tiempos.</p>
6	Pruebas	<p>Prueba el programa.</p> <p>Corrige y anota todos los errores encontrados.</p> <p>Anota el tiempo de pruebas en el Cuaderno de Registro de Tiempos.</p>
7	Postmortem	<p>Completa la tabla Resumen del Plan del Proyecto con los datos de tiempo, tamaño y defectos reales.</p> <p><i>Revisa los datos de defectos y actualiza la lista de comprobación para la revisión de código.</i></p> <p>Anota el tiempo postmortem en el Cuaderno de Registro de Tiempos.</p>
	Criterios de salida	<p>Programa probado a fondo.</p> <p>Diseño adecuadamente documentado.</p> <p><i>Lista de comprobación para la revisión de código completa.</i></p> <p>Listado completo del programa.</p> <p>Resumen del Plan del Proyecto completo.</p> <p>Cuadernos de Registro de tiempos y defectos completos.</p>

Tabla 14.5 Análisis de datos de defectos del Estudiante X.

Tipo	Introducido			Eliminado			Omitido
	Diseñar	Codificar	Otros	Revisar	Compilar	Pruebas	
10				4	4		4
20		8					
30							
40	2	3		1	4		4
50		2			1	1	2
60							
70							
80	2	3			1	4	5
90							
100							
Total	4	16		5	10	5	15
Programa							
10	2	6			6	2	8
11	1	5		3	2	1	3
12	1	5		2	2	2	4

Así, podemos hacer una comprobación **fácil** de todos los defectos contabilizados.

2. Clasifica los tipos de defectos en orden descendiente del número de defectos encontrados en las fases de compilación y pruebas. Un ejemplo de esta clasificación se muestra en la Tabla 14.6.
3. Para los tipos con el mayor número de defectos, examina el Cuaderno de Registro de Defectos y averigua los errores específicos que causan estos tipos. En la Tabla 14.6, tenemos: el tipo 80, defectos de función, el tipo 20 defectos sintácticos y el tipo 40 defectos de asignación.
4. Para los defectos que resultan de los problemas más importantes, idea los pasos necesarios en la revisión de código para encontrarlos. Supongamos que para el tipo 20 defectos de sintaxis, el Estudiante X encontró que su problema más frecuente era olvidar o colocar mal el punto y coma. Entonces decidió añadir una prueba que le recordase examinar cada línea del programa fuente para comprobar específicamente el punto y coma.
5. Registrar las comprobaciones ideadas en la Lista de Comprobación para la Revisión de Código para asegurarte que haces estos pasos. Por ejemplo, el Estudiante X pudo añadir el siguiente apartado punto y coma en la lista de comprobación: revisar cada línea del programa fuente para verificar que los punto y coma son utilizados de forma correcta.

Tabla 14.6 Clasificación de los datos de defectos del Estudiante X.

Tipo	Introducido			Eliminado			Omitido
	Diseñar	Codificar	Otros	Revisar	Compilar	Pruebas	
80	2	3			1	4	5
20		8		4	4		4
40	2	3		1	4		4
50		2			1	1	2
60							
100							
30							
10							
70							
90							
Total	4	16		5	10	5	15

6. Después de utilizar la nueva lista de comprobación, examina los datos de los defectos de la misma forma.
7. Si la lista de comprobación fue efectiva para encontrar esos defectos más importantes, añade otro tipo y utilízala de nuevo.
8. Si la lista de comprobación no fue efectiva para encontrar algunos tipos de defectos, cámbiala para detectar mejor estos defectos e intétalo de nuevo. En este caso, si el Estudiante X encontró que escribía con frecuencia dos puntos en vez de un punto y coma, pudo añadir una nota para comprobar cada punto y coma, y asegurarse de que no fue erróneamente escrito como dos puntos. Esta actualización de la lista de comprobación se muestra en la Tabla 14.7.
9. Al desarrollar o actualizar la lista de comprobación, agrupa las pruebas parecidas y no las duplique. Si una prueba no funciona bien, sustitúyela en vez de añadir una nueva prueba para la misma cosa. En el ejemplo de la Tabla 14.7, el Estudiante X incluyó las comprobaciones del punto y coma con otras comprobaciones de signos de puntuación.
10. Despues de desarrollar cada programa, examina los datos de defectos y la lista de comprobación para identificar los cambios o adiciones útiles.
11. Es una buena idea considerar qué pasos podrían prevenir estos defectos en el futuro. Como por ejemplo, la actualización de los estándares de codificación o añadir un paso al proceso de diseño.

En las Tablas 14.5 y 14.6, el Estudiante X mostró todos los defectos que había introducido y eliminado desde que comenzó a reunir datos de defectos. Aunque incluyó solamente 20 defectos, esos fueron todos los

Tabla 14.7 Lista de comprobación de Ada actualizada del Estudiante X.**Nombre del Programa y #:**

Propósito	Guía para realizar una revisión de código efectiva	#	#	#	#	Hasta la Fecha	% Hasta la Fech
Método	Cuando completes cada paso de la revisión, anota el número de defectos que has encontrado de cada tipo en la casilla de la derecha. Si no hay ninguno, anota un control en la casilla de la derecha. Completa la lista de comprobación para un programa, clase, objeto o método antes de comenzar a revisar la siguiente.						
Completo	Verifica que todas las funciones del diseño están programadas.	X					
Includes	Verifica que las sentencias <code>with</code> están completadas.	X					
Inicialización	Comprobar la inicialización de parámetros y variables: <ul style="list-style-type: none"> • Al inicio del programa. • Al comenzar cada bucle. • En la entrada a un procedimiento o función. 	X					
Llamadas	Comprobar los formatos de las llamadas a los procedimientos: <ul style="list-style-type: none"> • Signos de puntuación. • Parámetros. 	X					
Nombres	Comprobar la ortografía de los nombres y su utilización – <ul style="list-style-type: none"> • ¿Es consistente? • ¿Está dentro del ámbito declarado? • ¿Todas las estructuras y paquetes utilizan la referencia '.'? 	1				2	40
Cadenas de caracteres	Comprobar que todas las cadenas de caracteres hacen una utilización adecuada de la partición.	X					
Punteros	Comprobar los punteros: <ul style="list-style-type: none"> • Inicializado a <code>NULL</code>. • Solamente se eliminan después de <code>new</code>. • Siempre se eliminan después de utilizarlos. 	X					
Formato de salida	Comprobar el formato de salida: <ul style="list-style-type: none"> • ¿Es adecuado el salto de línea? • ¿Es adecuado el espaciado? 	X					
() Parejas	Asegurarse que los corchetes () son las adecuadas y están balanceados.	X					
Operadores lógicos	Verificar la utilización correcta de todos los operadores lógicos. Comprobar que cada función lógica tiene ().	X					
Comprobación línea a línea	Comprobar cada línea de código: <ul style="list-style-type: none"> • Sintaxis de la instrucción. • Utilización adecuada de los puntos y comas. • comprobar que los puntos y comas no se escriben como dos puntos. • Otros signos de puntuación. 	1				3	60
Estándares	Asegurar que cada programa se adapta a los estándares de codificación.	X					
Apertura y cierre de ficheros	Verifica que todos los ficheros son: <ul style="list-style-type: none"> • Declarados de forma adecuada. • Abiertos. • Cerrados. 	X					
Global	Hacer una revisión global al programa para comprobar los resultados del sistema y problemas inesperados.	X				5	100
Totales		2				5	100

datos que tenía. En la Tabla 14.5, puso primero los totales de los programas que había obtenido de los Resúmenes del Plan del Proyecto y después obtuvo del Cuaderno de Registro de Defectos información del tipo. En la Tabla 14.6, clasificó la Tabla 14.5 en orden decreciente al número de defectos que aparecen en la columna que está más a la derecha de la tabla. En la parte alta de la lista aparece el mayor número de defectos omitidos y a continuación el segundo tipo de defecto y así sucesivamente. Este tipo de clasificación se denomina distribución de Pareto. Estas distribuciones ordenan las entradas en un orden de prioridad, determinado por los datos. Obsérvese también, que puesto que el Estudiante X no hizo una revisión de código para el programa 10, contabilizó todos los defectos encontrados en la compilación y pruebas como defectos omitidos en la revisión de código.

14.5

LA MEJORA DE LA LISTA DE COMPROBACIÓN

Habítuate a revisar con frecuencia tus datos de defectos y vuelve a examinar la lista de comprobación. Cuando los pasos sean efectivos, vuelve sobre ellos. Cuando algún paso no sea adecuado, idea cómo puedes hacer el paso más efectivo y actualiza la lista de comprobación. Así, la lista de comprobación se convierte en un resumen de tu experiencia personal. También ayuda a seguir de forma consistente los pasos que personalmente has inventado para encontrar y corregir defectos. Los siguientes párrafos sugieren formas de mejorar tu lista de comprobación personal.

Después de terminar un programa, rellena la columna Hasta la Fecha de la lista de comprobación. Para ello, añade al valor Hasta la Fecha más reciente de dicha lista el número de defectos encontrados en cada paso de la revisión que acabas de completar. Actualiza el valor de la columna Hasta la Fecha para cada fila. El Estudiante X actualizó la lista de comprobación de la Tabla 14.7.

Rellena la columna de % Hasta la Fecha de esta forma: primero suma los valores de la columna Hasta la Fecha y anota el resultado en dicha columna y en la fila de total de la parte inferior de la lista de comprobación. Con dicho total, calcula el porcentaje para cada fila de Hasta la Fecha y escribe el resultado en la columna de % Hasta la Fecha. De nuevo, la Tabla 14.7 muestra un ejemplo.

Durante la fase postmortem para cada programa, compara la lista de comprobación con el cuaderno de registro de defectos para ver dónde y cómo, la lista de comprobación podría ser mejorada para perfeccionar el hallazgo de defectos. También debes considerar descartar los pasos de la revisión que no han encontrado o han omitido algún defecto entre los últimos cinco a diez programas más recientes. Aquí, por ejemplo, el

Tipos de defectos			
10 Documentación		50 Intelfaz	9 o Sistema
20 Sintaxis		60 Comprobación	10 o Entorno
30 Construcción Paquetes		70 Datos	
40 Asignación		80 Función	

Tabla 14.8 Ejemplo del cuaderno de registro de defectos del Estudiante X.

Estudiante	Estudiante X				Fecha	11/11/96
Profesor	Sr. Z				Programa #	12
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
11 / 1	1	20	codificac.	compilac.	1 min	
Descripción:		omitir:				
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
	2	20	codificac.	revisión	1 min	
Descripción:		escribir mal la variable X_axis como X_axes				
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
	3	20	diseño	compilac.	1 min	
Descripción:		; introducido como:				
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
	4	50	codificac.	compilac.	1 min	
Descripción:		incorrectamente escrita la llamada para el procedimiento Normalize				
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
	5	80	codificac.	pruebas	11 min	
Descripción:		olvidado inicializar las variables X_axis e Y_axis				
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
	6	80	diseño	pruebas	7 min	
Descripción:		bucle while no tiene un valor negativo para X_axis				
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
Descripción:						
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
Descripción:						

Estudiante X debería examinar el cuaderno de registro de defectos para el programa 12 en la Tabla 14.8 para ver si tiene que cambiar la lista de comprobación con objeto de mejorar la localización de defectos que ha omitido. Esta comprobación le convenció para añadir la referencia del punto y coma en la lista de comprobación de la Tabla 14.7.

Te sugiero que reúnas datos de más de 20 defectos antes de actualizar la lista de comprobación, no obstante, revisa los datos de defectos durante cada fase postmortem del programa. Cuando hayas visto los mismos defectos varias veces en la fase de compilación y en la de pruebas, considera la actualización de la lista de comprobación para tratar ese problema específico.

Reduce periódicamente la lista de comprobación. Dicha lista, por su naturaleza, crece con el tiempo. El poder de una lista de comprobación, sin embargo, es que centre la atención. Cuando crece mucho, se pierde la conciencia. Es importante revisar periódicamente los datos de defectos y eliminar las entradas de la lista de comprobación que no encuentren problemas. Puedes agrupar estas entradas eliminadas en una categoría de miscelánea para considerarlas cuando examines las otras entradas.

El método de la lista de comprobación personal reconoce que cada ingeniero es diferente y que las prácticas que unos ingenieros utilizan pueden no ser necesariamente efectivas para otros. Diseña tu propia lista de comprobación y examínala periódicamente para hacerla más efectiva. Conforme continúes omitiendo defectos en las revisiones de código, continúa buscando formas de mejorar la lista de comprobación. Recuerda, sin embargo, que las mejoras llegarán lentamente. Al principio, tu habilidad para encontrar defectos mejorará con cada revisión. Después, la mejora será **más** difícil. Reúne y analiza los datos de defectos y piensa qué podrías hacer para prevenir o mejorar la localización de defectos omitidos. Conforme vayas haciendo esto, continuarás mejorando las revisiones que hagas. También continuarás mejorando la calidad de los programas que hagas.

14.6

ESTÁNDARES DE CODIFICACIÓN

Una razón de por qué las listas de comprobación son efectivas es que proporcionan un estándar frente al cual se pueden revisar los programas. El principal estándar de revisión de código son las especificaciones sintácticas del lenguaje de programación, pero estas no especifican estilos o formatos de codificación. Por esto, necesitas un estándar de codificación.

Un **estándar** es una base para la comparación oficialmente aceptada. Un **estándar de codificación** define un conjunto de prácticas de codificación aceptadas, las cuales pueden servir como un modelo para tu trabajo. Este estándar deberías utilizarlo como una guía cuando escribas código

fuente. Dichos estándares, normalmente especificarán la forma en la que el código fuente es formateado, qué sentencias van en líneas de texto separadas, y cómo se sangran las sentencias. Se definen prácticas para escribir comentarios, incluyendo cuándo son necesarios los comentarios explicativos. Normalmente, el nombre del ingeniero, la fecha de trabajo, el nombre del programa, el nombre del proyecto y la versión se ponen en una cabecera de comentarios en la parte superior del listado del programa. Un ejemplo del estándar de codificación utilizado con el lenguaje C++ se muestra en la Tabla 14.9.

Los estándares de codificación también pueden ser útiles para prevenir defectos. Aquí, por ejemplo, puedes utilizar ciertas prácticas para evitar, sentencias como go-to, tener múltiples salidas en los procedimientos o utilizar rutinas recursivas. Algunas prácticas también son generalmente muy útiles, tales como inicializar siempre las variables a la entrada al bucle o cuando las declares. Las prácticas de poner nombres inadecuados pueden ser la principal fuente de error. Se deben utilizar nombres que claramente se relacionen con las funciones de las variables, y los nombres deberían ser lo bastante diferentes para que no se confundiesen fácilmente. Dos nombres que se confunden y son propensos a error serían XY34B y XY35C. Una elección más adecuada sería AZIM34 y ACTUALIZA–AZIM.

Si tu profesor ha establecido un estándar de codificación, consigue una copia y utilízala. Te ayudará a hacer programas más legibles y fácilmente comprensibles. La legibilidad del código también ayuda en las pruebas y en la depuración de los programas, y ayudará a cualquiera que quiera utilizar o modificar tus programas.

RESUMEN

Este capítulo introduce las listas de comprobación para la revisión de código y describe cómo puedes desarrollarlas y utilizarlas. Una lista de comprobación contiene una serie de pasos que tú quieras seguir de forma rigurosa. Cuando utilizas una lista de comprobación desarrollada a partir de tus propios defectos, harás revisiones más eficientes. La lista de comprobación no solamente ayuda a encontrar más defectos, también ayuda a encontrarlos más rápidamente.

Para construir una lista de comprobación para la revisión de código, adáptala al lenguaje que utilices, diséñala a partir de tus propios defectos y ajústala a tus habilidades y experiencia cambiante.

Algunas orientaciones para utilizar la lista de comprobación son: haz las revisiones paso a paso, completa cada programa o procedimiento antes de iniciar el siguiente, examina cada apartado de la lista de comprobación cuando lo completes. Cuando encuentres defectos, anota el

Tabla 14.9 Estándar de codificación de C++.

Propósito	Guia el desarrollo de programas en C++.
Cabeceras del programa	Todos los programas comienzan con una cabecera descriptiva
Formato de la cabecera	<pre>***** /* Programa ejercicio: el número de programa */ /* Nombre: tu nombre */ /* Fecha: fecha de inicio de */ /* desarrollo del programa */ /* Descripción: una breve descripción de */ /* la función del programa */ *****</pre>
Listado de contenidos	Proporciona un resumen de la lista de contenidos
Ejemplos de contenidos	<pre>***** /* Listado de contenidos: * Instrucciones a reutilizar * Includes * Declaraciones de clases: * CData * ASet * Código fuente en c:\clases\CData.cpp: * CData * CData() * Empty() *****</pre>
Instrucciones de reutilización	<p>Describe cómo se utiliza el programa. Proporciona el formato de declaración, valores de los parámetros y tipos, y límites de los parámetros.</p> <p>Proporciona avisos para valores no válidos, condiciones de desbordamiento u otras condiciones que podrían, en potencia, dar lugar a operaciones inadecuadas.</p>
Ejemplo	<pre>***** /* Instrucciones de reutilización * int PrintLine (char *linea_de_caracteres) * Propósito: imprimir cadena de caracteres, * `linea_de_caracteres`, o imprimir una * línea * Limitaciones: la longitud máxima de la * línea es LONGITUD-LINEA * Devuelve: 0 si la impresora no está * preparada para imprimir sino 1 *****</pre>
Identificadores	Utiliza nombres descriptivos para todas las variables, nombres de funciones, constantes y otros identificadores. Evita abreviaturas o variables de una sola letra.
Ejemplo de identificador	<pre>int numero_de_estudiantes; /*Está bien*/ float x4, j, ftave; /*Está mal*/</pre>

(Continúa)

Tabla 14.9 Estándar de codificación de C++.
(Continuación)

Comentarios	Documenta el programa suficientemente para que el lector pueda entenderlo. Los comentarios deberían explicar tanto el propósito como el funcionamiento del código. Comenta la declaración de variables indicando su propósito.
Buen comentario	If(contador_reg > limite) /* ¿Han sido /* procesados todos los registros ? */
Mal comentario	If(contador_reg > limite) /* comprueba si /* contador-reg es mayor que limite */
Secciones principales	Las secciones principales del programa deberían ser precedidas por un bloque de comentarios que describan el procesamiento que se realizará en la sección siguiente.
Ejemplo	/***** /* Esta sección del programa examinará los /* contenidos del array "notas" y calculará /* la nota media de la clase *****/
Espacios en blanco	Escribe programas con suficientes espacios en blanco para facilitar la lectura. Separa cada bloque con al menos un espacio.
Sangrado	Sangra cada nivel de corchete con respeto al anterior. La apertura y cierre de corchetes deberían hacerse uno por línea estar alineados unos con respecto a otros.
Ejemplo de sangrado	while (distancia-que-falta > umbral) { exito_codigo = mueve-robot (localización_objetivo); if (exito-código == MOVIMIENTO-FALLIDO) { printf("Ha fallado el movimiento del robot"); } }
Mayúsculas	Todos los defines se escriben en mayúsculas. Todos los otros identificadores y palabras reservadas en minúscula. En los mensajes de usuario pueden ir mezcladas las mayúsculas y minúsculas, para que la presentación sea más clara.
Ejemplos de mayúsculas	#define NUMERO-ESTUDIANTES-POR-DEFECTO 15 int tamaño-clase= NUMERO_ESTUDIANTES_POR_DEFECTO;

número encontrado en cada apartado de la lista de comprobación. Cuando lo hagas, completa las columnas Hasta la Fecha y % Hasta la Fecha. Después de acabar cada programa, revisa los datos y la lista de comprobación para ver cómo la puedes mejorar.

EJERCICIO 14

Haz una lista de comprobación para encontrar los defectos que causan la mayor parte de problemas durante la compilación y las pruebas. Restringe esta lista de comprobación inicial a unos pocos tipos de defectos y utilízala en tu próximo programa. Entrega una lista de comprobación acabada, junto con el programa y completa el Resumen del Plan del Proyecto. Entrega una copia de los datos de los defectos utilizados para hacer la lista de comprobación. Para analizar los defectos, utiliza un formato similar al mostrado en la Tabla 14.6.

Entrega copias del Cuaderno de Registro de Tiempos y de las hojas del Resumen Semanal de Actividades que no hayas entregado previamente. También, entrega copias completas del Resumen del Plan del Proyecto, del Cuaderno de Registro de Defectos y de la Lista de Comprobación para la Revisión de Código para cada programa que desarrolles. Incluye los tiempos de desarrollo planificados y reales y los defectos reales introducidos y eliminados.

CAPITULO 15

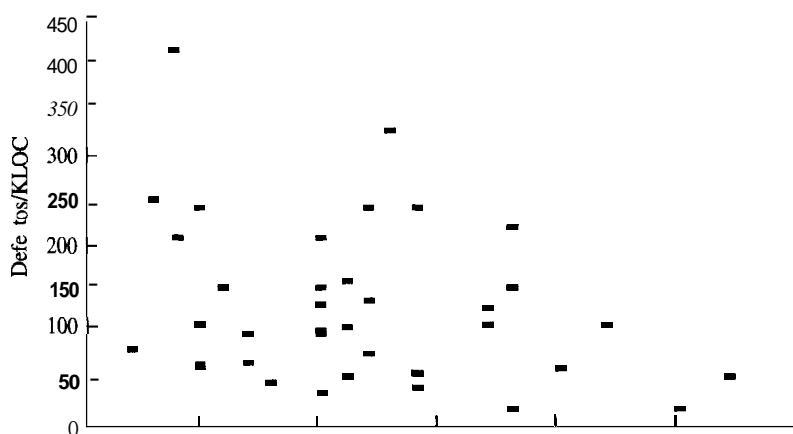
La previsión de defectos

Este capítulo discute las formas de analizar y utilizar tus datos de defectos para ayudarte a mejorar tanto la precisión de la planificación como la calidad del producto. También, se muestran ejemplos útiles de cómo analizar estos datos de defectos. Como ejercicio, prepararás un breve resumen de los datos de defectos para los programas que hayas desarrollado hasta la fecha en este curso.

La razón principal para utilizar **los** datos de defectos, es para determinar cómo mejorar la prevención o localización de defectos que hayas introducido. En los últimos capítulos, introducimos medidas que ayudan a controlar la calidad de tu trabajo. Esto te ayudará a que de una forma más consistente produzcas programas de alta calidad.

15.1 EL PORCENTAJE DE DEFECTOS

El porcentaje de defectos introducidos por los ingenieros del software experimentados, oscila normalmente en un rango de entre 50 a 250 defectos/KLOC. La Figura 15.1 muestra los defectos/KLOC que un grupo de 38 ingenieros introdujo en un ejercicio de codificación pequeño. Esto fue antes de que ellos comenzaran a medir, revisar y controlar el número de defectos que introducían. Esta figura muestra el porcentaje de defectos introducidos por estos ingenieros antes de que fuesen entrenados en el PSP.



Obsérvese también que la Figura 15.1 muestra la densidad de defectos en función de los años de experiencia del ingeniero en escribir programas. Algunos de los ingenieros menos experimentados tenían un número más elevado de defectos que los otros, sin embargo para estos pocos ingenieros, los años de experiencia tenían poca influencia en el número de defectos. Esto implica que hay algunas prácticas de calidad, que muchos ingenieros aprenden con la experiencia. Algunos adquirían la experiencia más rápidamente que otros. Entonces, a partir de un grado de experiencia similar, los defectos dependen en gran parte de la disciplina personal.

La Figura 15.2 muestra los datos de los mismos 38 ingenieros después de haber completado un curso del PSP [Humphrey 95]. En este punto, habían revisado sus defectos para un total de 10 programas pequeños, y habían hecho dos estudios de los tipos, distribuciones y tiempos de corregir sus defectos. Como resultado, comprendieron el número y tipos de defectos que normalmente introducían y cuánto tiempo le dedicaban a corregirlos. Esto les ayuda a valorar la importancia de las prácticas personales disciplinadas, y les motivó a ser más cuidadosos en el diseño y codificación de sus programas. Como resultado, también redujeron su porcentaje de introducción de defectos. Los datos de las Figuras 15.1 y 15.2 sugieren que la disciplina personal, asociada con la revisión de defectos y su análisis, es mucho más efectiva que los años de experiencia para la reducción del número de defectos introducidos.

Otro mensaje de las Figuras 15.1 y 15.2 es que los ingenieros del software introducen muchos defectos. Con pocas excepciones, la tasa de introducción de defectos por grupos de software está alrededor de los 100 defectos/KLOC o más. Incluso los ingenieros que han aprendido a ges-

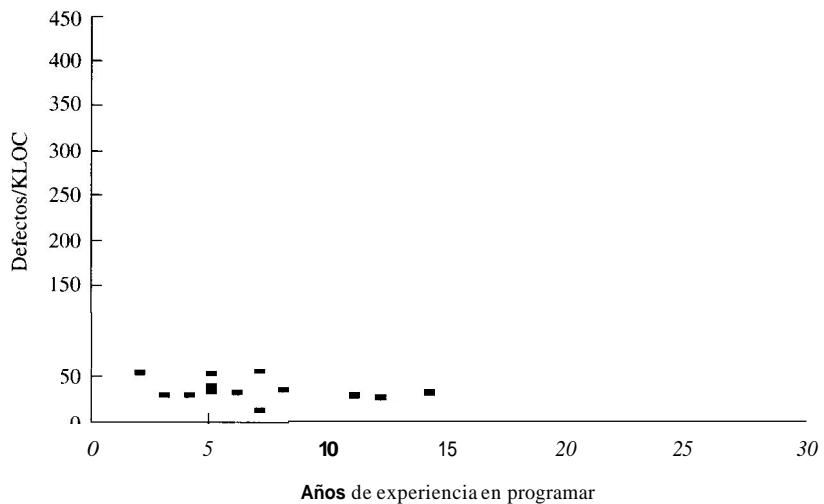


Figura 15.2 Defectos/KLOC vs. años - decpues del entrenamiento en el PSP.

tionar los defectos, introducen una media de unos 50 defectos/KLOC. Es decir 1 defecto por cada 20 LOC. Aunque esto no pueda parecer una gran cantidad, serían 50 defectos en un programa de 1.000 LOC, 500 en uno de 10.000 LOC y 50.000 en uno de 1.000.000 LOC. Todos estos defectos se deben localizar y corregir antes de que puedas estar seguro de que el programa funcionará adecuadamente. Repitiendo una cuestión clave, puesto que la introducción de defectos es una consecuencia natural del ser humano, todos los ingenieros del software introducen defectos. Todos los ingenieros deberían por lo tanto comprender el número y tipo de defecto que introducen.

15.2 LA UTILIZACIÓN DE LOS DATOS DE DEFECTOS

Hasta ahora, has reunido datos de defectos para ayudarte a comprender los defectos que has introducido. A continuación utilizaste esos datos para diseñar una lista de comprobación personal para hacer la revisión de código. En este capítulo, utilizarás esos datos de defectos para estimar el número de defectos que introducirás en un nuevo programa. Utilizando datos históricos, también verás cómo haces unas buenas estimaciones del número de defectos que introducirás y eliminarás en cada fase de un proyecto de codificación.

Las estimaciones exactas del grado de defectos son importantes. Siempre puedes realizar otra prueba u otra revisión de código. La única manera para decidir realizar dichas pruebas y revisiones, consiste en analizar los datos de los defectos. Comparando los datos del proyecto en cur-

so con los datos históricos, puedes determinar la posible calidad del programa a desarrollar. Puedes decidir si son necesarios pasos adicionales para la eliminación de los defectos.

Todas estas medidas y planes no te ayudarán mucho, a menos que estés personalmente comprometido a hacer programas libres de defectos. Aunque es costoso producir programas de alta calidad y rara vez serás capaz de hacerlo al primer intento, con mucha práctica, harás cada vez más programas libres de defectos. Esto es importante, puesto que algún día probablemente trabajarás en un proyecto, donde es esencial el trabajo libre de defectos. Te sugiero que consideres cada programa que desarrollas como una práctica para ese día.

15.3 DENSIDAD DE DEFECTOS

La unidad de medida de defectos utilizada en el PSP es la de defectos por miles de líneas de código (KLOC). Esto se denomina densidad de defectos (Dd) y se mide en unidades de defectos/KLOC. Aquí, la K hace referencia a 1.000. Para calcular el total de defectos/KLOC encontrados en un programa, haz lo siguiente:

1. Suma el total de número de defectos (D) encontrados en todas las fases del proceso.
2. Cuenta el número (N) de líneas de código nuevas o cambiadas en un programa.
3. Calcula la densidad de defectos por KLOC como $Dd = 1.000 * D/N$.

Por ejemplo, si un programa de 96 LOC tenía un total de 14 defectos, entonces la densidad de defectos sería $1.000 * 14 / 96 = 145,83$ defectos/KLOC.

En el cálculo de las densidades de defectos, es importante utilizar medidas de tamaño correctas. Tal y como se describió en el Capítulo 6, este libro cuenta el tamaño del programa por el número de LOC nuevas y cambiadas. Es decir, si utilizas una rutina de librería o copias una parte de un programa, esto no se contabiliza como LOC. Sin embargo, contabiliza como LOC nueva, cualquier LOC modificada de los programas que habías copiado.

15.4 CÓMO ESTIMAR LAS TASAS DE DEFECTOS

Cuando estés desarrollando un nuevo programa, probablemente tendrás problemas en estimar cuántos defectos introducirás. La razón es que este número variará de un programa al siguiente. Hay varias causas para esto.

Primera, con la experiencia, tus habilidades mejorarán. Cuando comienzas a programar, te enfrentas a muchos problemas que no te habías encontrado anteriormente. No puedes estar seguro de cómo trabaja alguna función o procedimiento, puedes confundirte con algún elemento del lenguaje, o puedes encontrar nuevos aspectos del entorno o del compilador. Estos problemas darán lugar a que tus tiempos de desarrollo y el porcentaje de introducción de defectos fluctúe. Con la experiencia, superarás de forma gradual estos problemas iniciales y cometerás pocos errores. Esto reducirá tanto el número total de defectos como la variación de estos números. La reducción inicial en el número de defectos será como resultado de la mayor experiencia y de tu mejor desenvoltura con el lenguaje. **Más** allá de esta mejora inicial, sin embargo, necesitarás reunir y analizar **los** datos de los defectos para continuar mejorando.

La segunda razón para la fluctuación en la tasa de defectos, es que tu proceso no es estable. Es decir, conforme aprendas a escribir programas, también aprenderás nuevos métodos y procedimientos. Tus prácticas de trabajo evolucionarán. Esto causará fluctuaciones en el tiempo que necesitas para hacer las distintas tareas de codificación y en el número de los defectos introducidos. Finalmente, **los** mismos defectos son una fuente de fluctuación. A mayor número de defectos introducidos, será necesario más tiempo para repararlos. Puesto que los ingenieros normalmente introducen de 6 a **8** defectos por hora durante la codificación de un programa, cada hora dedicada a la corrección de defectos aumentará la probabilidad de introducir más defectos. Puesto que el tiempo de corrección de defectos es generalmente variable. un proceso que introduce muchos defectos es intrínsecamente impredecible.

Conforme vayas mejorando **los** procesos, sin embargo, te estabilizarás y esta estabilidad mejorará la precisión de tus previsiones de defectos. Si dedicas una cantidad de tiempo suficiente a las revisiones de código, tus procesos tenderán a estabilizarse a un modelo fijo. Una vez que tus procesos sean razonablemente estables, serán más predecibles. **Así**, revisando el número de defectos/KLOC introducidos y eliminados en los **últimos** programas, puedes estimar con precisión el número de defectos que probablemente introducirás y eliminarás en el futuro.

15.5 ESTIMACIÓN DE DEFECTOS

Cuando estés planificando un nuevo programa, estima primero el número de LOC nuevas y cambiadas que probablemente tendrá el programa. A continuación, calcula el valor medio de los defectos/KLOC de los programas que hayas desarrollado anteriormente. Con estos datos, ahora, puedes calcular el número de defectos/KLOC esperados en el nuevo programa:

$$Dd_{\text{plan}} = 1.000(D_1 + \dots + D_i)/(N_1 + \dots + N_i)$$

Supongamos, por ejemplo, que tenías datos de los 5 programas mostrados en la Tabla 15.1. El valor de Dd_{plan} se calcula de esta forma:

$$\begin{aligned} Dd_{\text{plan}} &= 1.000(6 + 11 + 7 + 9 + 5)/(37 + 62 + 49 + 53 + 28) \\ &= 1.000 * 38 / 229 = 165,94 \text{ defectos/KLOC} \end{aligned}$$

Suponiendo que el nuevo programa tendrá la misma densidad de defectos, el número de defectos esperados será:

$$D_{\text{plan}} = N_{\text{plan}} * Dd_{\text{plan}} / 1.000.$$

Ahora, utilizando este mismo ejemplo, y suponiendo que las LOC estimadas para el nuevo programa son 56, el número de defectos esperados es:

$$\begin{aligned} D_{\text{plan}} &= 56 * 165,94 / 1.000 \\ &= 9,29 \text{ defectos.} \end{aligned}$$

Utilizando estos datos, tendrás una previsión de 9 defectos para un proyecto de codificación con 56 LOC planificadas.

El tamaño Hasta la Fecha y los datos de defectos en el Resumen del Plan del Proyecto fueron diseñados para ayudarte a hacer estos cálculos.

$$D_{\text{plan}} = N_{\text{plan}} * D_{\text{Hasta la Fecha}} / N_{\text{Hasta la Fecha}},$$

O, utilizando los datos de la Tabla 15.1, esta ecuación da:

$$D_{\text{plan}} = 56 * 38 / 229 = 9,29,$$

obteniéndose el mismo resultado que antes.

Con el número total de defectos esperados para el nuevo programa, puedes calcular el número de defectos esperados que se introducirán y eliminarán en cada fase. Para obtener estos números, multiplica el número total de defectos esperados por el valor de % Hasta la Fecha para cada fase y divídelo por 100. Esto nos da el número de defectos esperados para esa fase del nuevo proyecto. Esta fue la razón de calcular los valores de los defectos Hasta la Fecha y el % de defectos Hasta la Fecha. Ya que,

Tabla 15.1 Ejemplo de datos de defectos.

Número de programa	Defectos (D)	LOC (N)
1	6	37
2	11	62
3	7	49
4	9	53
5	5	28
Total Hasta la Fecha	38	229

proporcionan los datos históricos necesarios para estimar los defectos. En la Sección 15.7 se muestra un ejemplo de estos cálculos con la actualización del Resumen del Plan del Proyecto.

15.6

ACTUALIZACIÓN Y EJEMPLO DE LA TABLA RESUMEN DEL PLAN DEL PROYECTO

El nuevo guión del Proceso PSP y la plantilla del Resumen del Plan del Proyecto incluyen ahora la planificación de defectos, lo podemos observar en las Tablas 15.2 y 15.3. Las instrucciones para el Resumen del Plan del Proyecto se muestran en la Tabla 15.4. Las nuevas entradas de estas tablas se muestran en **cursiva**. Las nuevas incorporaciones son: la columna del Plan para los defectos introducidos y eliminados y la fila defectos/KLOC en la sección resumen.

La Tabla 15.5 muestra un ejemplo de la tabla Resumen del Plan del Proyecto del Estudiante X para el programa 13. Los siguientes párrafos describen cómo ha utilizado los datos del programa 12, resumidos en la Tabla 15.6, para completar los datos de la Tabla 15.5. Varios de los valores de esta tabla están etiquetados con letras para ayudar a identificar la fuente del dato. Primero, durante la planificación:

1. Primero el Estudiante X estimó el tamaño del nuevo programa en 58 LOC (a) y el máximo y mínimo en 72 (a) y 41 (a) LOC.
2. Después, observó la tabla Resumen del Plan del Proyecto del programa 12 para encontrar los minutos/LOC Hasta la Fecha. Como se observa, este valor era 5,92 (b) minutos/LOC. Utilizó dicho valor como velocidad de planificación para el programa 13.
3. Con el total estimado de 58 LOC (a), el tiempo total estimado del proyecto es: $58 \times 5,92 = 343,36$ o 343 minutos (c).
4. Entonces el Estudiante X calculó los tiempos de desarrollo máximo y mínimo, multiplicando los tamaños máximo y mínimo (a) por 5,92 para obtener 426 y 243 (d) minutos, respectivamente.
5. Para el tiempo por fase, tomó los % Hasta la Fecha del resumen (e) del programa 12 y los multiplicó por el tiempo total estimado de 343 minutos (c), y dividió el resultado por 100. Esto dio los tiempos para cada fase, tal y como se muestra en (f). Por comodidad, redondeó los minutos a un valor entero.
6. Para el total de defectos introducidos y eliminados, el Estudiante X encontró el valor de 94,79 (g) defectos/KLOC Hasta la Fecha en la Tabla 15.6 del Resumen del Plan del Proyecto para el programa 12. Como se ve abajo, este valor se calculó a partir de LOC Hasta la Fecha (h) y los defectos Hasta la Fecha (i).

Tabla 15.2 Guión del proceso del PSP.

Propósito	Guiaerte en el desarrollo de pequeños programas.
Entradas requeridas	<p>La descripción del problema.</p> <p>Tabla Resumen del Plan del Proyecto PSP.</p> <p>Una copia de la lista de comprobación para la revisión de código.</p> <p>Datos de tamaños y tiempos reales de programas anteriores.</p> <p>Cuaderno de Registro de Tiempos.</p> <p>Cuaderno de Registro de Defectos.</p>
1 Planificación	<p>Obtén una descripción de las funciones del programa.</p> <p>Estima las LOC Máx., Mín., total requeridas.</p> <p>Determina los Minutos/LOC.</p> <p>Calcula los tiempos de desarrollo Máx., Mín., y total.</p> <p>Estima los defectos a introducir y eliminar en cada fase.</p> <p>Escribe los datos del plan en la tabla Resumen del Plan del Proyecto.</p> <p>Anota el tiempo de planificación en el Cuaderno de Registro de Tiempos.</p>
2 Diseño	<p>Diseña el programa.</p> <p>Anota el diseño en el formato especificado.</p> <p>Anota el tiempo de diseño en el Cuaderno de Registro de Tiempos.</p>
3 Codificación	<p>Implementa el diseño.</p> <p>Utiliza un formato estándar para introducir el código.</p> <p>Anota el tiempo de codificación en el Cuaderno de Registro de Tiempos.</p>
4 Revisión de código	<p>Revisar completamente el código fuente.</p> <p>Seguir el guión de revisión de código de la lista de comprobación.</p> <p>Corregir y registrar todos los defectos encontrados.</p> <p>Registrar el tiempo de revisión en el Cuaderno de Registro de Tiempos.</p>
5 Compilación	<p>Compila el programa.</p> <p>Corrige y registra todos los errores encontrados.</p> <p>Anota el tiempo de compilación en el Cuaderno de Registro de Tiempos.</p>
6 Pruebas	<p>Prueba el programa.</p> <p>Corrige y anota todos los errores encontrados.</p> <p>Anota el tiempo de pruebas en el Cuaderno de Registro de Tiempos.</p>
7 Postmortem	<p>Completa la tabla Resumen del Plan del Proyecto con los datos de tiempo, tamaño y defectos reales.</p> <p>Revisa los datos de defectos y actualiza la lista de comprobación para la revisión de código.</p> <p>Anota el tiempo postmortem en el Cuaderno de Registro de Tiempos.</p>
Criterios de salida	<p>Programa probado a fondo.</p> <p>Diseño adecuadamente documentado.</p> <p>Lista de comprobación para la revisión de código completa</p> <p>Listado completo del programa.</p> <p>Resumen del Plan del Proyecto completo.</p> <p>Cuaderno de Registro de tiempos y defectos completos.</p>

Tabla 15.3 Resumen del plan del proyecto del PSP.

Estudiante _____	Fecha _____		
Programa _____	Programa # _____		
Profesor _____	Lenguaje _____		
Resumen	Plan	Real	Hasta la Fecha
Minutos/LOC	_____	_____	_____
LOC/Hora	_____	_____	_____
Defectos/KLOC	_____	_____	_____
Rendimiento	_____	_____	_____
V/F	_____	_____	_____
Tamaño Programa (LOC):			
Total Nuevo & Cambiado	_____	_____	_____
Tamaño Máximo	_____	_____	_____
Tamaño Mínimo	_____	_____	_____
Tiempo por Fase (min.)	Plan	Real	Hasta la Fecha
Planificación	_____	_____	_____
Diseño	_____	_____	_____
Codificación	_____	_____	_____
Revisión del código	_____	_____	_____
Compilación	_____	_____	_____
Pruebas	_____	_____	_____
Postmorten	_____	_____	_____
Total	_____	_____	_____
Tiempo Máximo	_____		
Tiempo Mínimo	_____		
Defectos introducidos	Plan	Real	Hasta la Fecha
Planificación	_____	_____	_____
Diseño	_____	_____	_____
Codificación	_____	_____	_____
Revisión del código	_____	_____	_____
Compilación	_____	_____	_____
Pruebas	_____	_____	_____
Total	_____	_____	_____
Defectos eliminados	Plan	Real	Hasta la Fecha
Planificación	_____	_____	_____
Diseño	_____	_____	_____
Codificación	_____	_____	_____
Revisión del código	_____	_____	_____
Compilación	_____	_____	_____
Pruebas	_____	_____	_____
Total	_____	_____	_____

7. Con los valores de 94,79 defectos/KLOC (i) y las 58 LOC (a) planificadas, calculó el número total de defectos esperados $94,79 \times 58 / 1.000 = 5,50$, aproximadamente 6 defectos (j).

8. Basándose en los valores de % Hasta la Fecha para los defectos introducidos y eliminados del programa 12 (k), calculó la estimación del número de defectos a ser introducidos y eliminados por fase (l).

Tabla 15.4 Instrucciones del resumen del plan del proyecto del PSP.

Propósito	Esta tabla trata los datos estimados y reales de los proyectos de una forma cómoda y fácilmente recuperable.
Cabecera	Introduce los siguientes datos: <ul style="list-style-type: none"> • Tu nombre y fecha de hoy. • Nombre y número de programa. • Nombre del profesor. • El lenguaje que utilizarás para escribir el programa.
Minutos/LOC	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> • Escribe los Minutos/LOC planificados para este proyecto. Utiliza la velocidad Hasta la Fecha de un programa reciente del Cuaderno de Trabajos o del Resumen del Plan del Proyecto. Después del desarrollo: <ul style="list-style-type: none"> • Divide el tiempo total de desarrollo por el tamaño real del programa para obtener los Minutos/LOC reales y los Minutos/LOC Hasta la Fecha. • Por ejemplo, si el proyecto se hizo en 196 minutos e hiciste 29 LOC, los Minutos/LOC serían $196/29 = 6,76$.
LOC/Hora	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> • Calcula las LOC por hora planificada para este programa dividiendo 60 por los Minutos/LOC de la casilla de Flan. Después del desarrollo: <ul style="list-style-type: none"> • Para LOC/Hora Real y Hasta la Fecha divide 60 por Minutos/LOC Reales y Hasta la Fecha. • Para los 6,76 Minutos/LOC Reales, tenemos $60/6,76 = 8,88$ LOC/Hora Reales.
Defectos/KLOC	Antes de desarrollar: <ul style="list-style-type: none"> • Encuentra los defectos/KLOC Hasta la Fecha del programa más reciente. • Utiliza dicho valor como los defectos/KLOC planificados. Después del desarrollo: <ul style="list-style-type: none"> • Calcula los valores de defectos/KLOC reales y Hasta la Fecha para este programa. • Para el valor real, multiplica el total de defectos reales por 1.000 y divídelo por las LOC Total Nuevas & Cambiadas Reales. • Para el valor Hasta la Fecha haz unos cálculos similares. • Con 17 defectos Hasta la fecha y 153 LOC Total Nuevas & Cambiadas, los defectos/KLOC Hasta la Fecha = $1.000 * 17 / 153 = 111,11$.
Tamaño Programa (LOC)	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> • Escribe bajo la columna plan, el valor estimado de LOC Total Nuevas & Cambiadas, Máximo y Mínimo. Después del desarrollo: <ul style="list-style-type: none"> • Cuenta y anota las LOC Nuevas & Cambiadas Reales. • Para la columna Hasta la Fecha, añade LOC reales Nuevas & Cambiadas a las LOC Hasta la Fecha Nuevas & Cambiadas de programas anteriores.

(Continúa)

Tabla 15.4 Instrucciones del resumen del plan del proyecto del PSP.
(Continuación)

Tiempo por fase	
Plan	<p>Para el tiempo total de desarrollo, multiplica el valor de LOC Total Nuevas & Cambiadas por los Minutos/LOC.</p> <p>Para el tiempo Máximo, multiplica el tamaño Máximo por los Minutos/LOC.</p> <p>Para el tiempo Mínimo, multiplica el tamaño Mínimo por los Minutos/LOC.</p> <p>Del Resumen del Plan del Proyecto de un programa reciente, busca los valores de % Hasta la Fecha para cada fase.</p> <p>Utilizando el % Hasta la Fecha de programas anteriores calcula el tiempo planificado para cada fase.</p>
Real	<p>Una vez acabado el trabajo, anota el tiempo real en minutos que has gastado en cada fase del desarrollo.</p> <p>Obtén estos datos del Cuaderno de Registro de Tiempos.</p>
Hasta la Fecha	Para cada fase, escribe la suma del tiempo real y el tiempo Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, escribe 100 multiplicado por el tiempo Hasta la Fecha y lo divides por el total del tiempo Hasta la Fecha.
Defectos introducidos	
Plan	<p><i>Antes del desarrollo, estima el número total de defectos a introducir en el programa.</i></p> <p><i>El valor es: defectos/KLOC planificados multiplicado por las LOC Total Nuevas & Cambiadas planificadas para este programa y dividido por 7.000.</i></p> <p><i>Por ejemplo, con un valor de defectos/KLOC planificado de 75,9 y un valor de LOC Nuevas & Cambiadas planificadas de 75, el total de Defectos Planificados = 75,9*75/1.000=5,69, redondeado a 6.</i></p> <p><i>Antes de desarrollar, estima los defectos introducidos por fase utilizando el total de defectos estimados y el % Hasta la Fecha de defectos introducidos de programas anteriores.</i></p>
Real	Después del desarrollo, localiza y anota el número real de defectos introducidos en cada fase.
Hasta la Fecha	Para cada fase, escribe la suma de los defectos reales y los defectos Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, multiplica por 100 los defectos Hasta la Fecha para esa fase y divídilos por el total de defectos Hasta la Fecha.
Defectos eliminados	
Plan	<p><i>En la fila de total, escribe el total de defectos estimados.</i></p> <p><i>Utilizando los valores de % Hasta la Fecha de los programas más recientes, calcula los defectos eliminados planificados para cada fase.</i></p>
Real	Después del desarrollo, localiza y anota el número real de defectos eliminados en cada fase.
Hasta la Fecha	Para cada fase, escribe la suma de los defectos reales y los defectos Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, multiplica por 100 los defectos Hasta la Fecha para esa fase y divídilos por el total de defectos Hasta la Fecha.

Tabla 15.5 Ejemplo de resumen del plan del proyecto del PSP.

Estudiante	Estudiante X			Fecha	18/11/96
Programa				Programa#	13
Profesor	Sr. Z			Lenguaje	Ada
Resumen		Plan	Real	Hasta la Fecha	
Minutos/LOC	b	5,92	P	4,87	x
LOC/Hora		10,14	q	12,32	y
Defectos/KLOC	i	94,79	r	106,4	z
Rendimiento					
V/F					
Tamaño Programa (LOC):					
Total Nuevo & Cambiado	a	50	o	47	w
Tamaño Máximo	a	72			
Tamaño Mínimo	a	41			
Tiempo por Fase (min.)		Plan	Real	Hasta la Fecha	% Hasta la Fecha
Planificación	f	18 m	22 s	88 u	6,0
Diseño	f	35 m	24 s	151 u	10,2
Codificación	f	149 m	93 s	637 u	43,1
Revisión del código	f	20 m	37 s	111 u	7,5
Compilación	f	24 m	4 s	92 u	6,2
Pruebas	f	64 m	8 s	240 u	16,2
Postmorten	f	33 m	41 s	160 u	10,8
Total	c	343 m	229 s	1479 u	100
Tiempo Máximo	d	426			
Tiempo Mínimo	d	243			
Defectos introducidos		Plan	Real	Hasta la Fecha	% Hasta la Fecha
Planificación	j	1	t	4 u	16,0
Diseño	j	5 n	5 t	21 u	84,0
Codificación					
Revisión del código					
Compilación					
Pruebas					
Total	j	6 n	5 t	25 u	100,0
Defectos eliminados		Plan	Real	Hasta la Fecha	% Hasta la Fecha
Planificación	j	2 n	3 t	8 u	32,0
Diseño	j	3 n	2 t	12 u	48,0
Codificación					
Revisión del código					
Compilación					
Pruebas	j	1	t	5 u	20,0
Total	j	6 n	5 t	25 u	100,0

Esto ahora completa las estimaciones de defectos para el programa 13.

15.7

LA ANOTACIÓN DE LOS DATOS REALES

Después de desarrollar el programa, el Estudiante X anotó los datos reales en la tabla, de la siguiente forma:

Tabla 15.6 Resumen del plan de programa 12 del Estudiante X.

Estudiante	Estudiante X	Fecha	11/11/96		
Programa		Programa#	12		
Profesor	Sr. Z	Lenguaje	Ada		
Resumen	Plan	Real	Hasta la Fecha		
Minutos/LOC	6,30	4,93 b	5,92		
LOC/Hora	9,52	12,17	10,14		
Defectos/KLOC		b	94,79		
Rendimiento					
V/F					
Tamaño Programa(LOC):					
Total Nuevo & Cambiado	51	58 h	211		
Tamaño Máximo	65				
Tamaño Mínimo	37				
Tiempo por fase (min.)	Plan	Real	Hasta la Fecha	% Hasta la Fecha	
Planificación	16	18 s	66 e	5,3	
Diseño	27	44 s	127 e	10,2	
Codificación	146	104 b	544 e	43,5	
Revisión del código	12	38 s	74 e	5,9	
Compilación	26	11 s	88 e	7,0	
Pruebas	68	29 s	232 e	18,6	
Postmorten	26	42 s	119 e	9,5	
Total	321	286 b	1250	100,0	
Tiempo Máximo	410				
Tiempo Mínimo	233				
Defectos introducidos	Plan	Real	Hasta la Fecha	% Hasta la Fecha	Def./Hora
Planificación					
Diseño		1 t	4 k	20,0	
Codificación		5 t	16 k	80,0	
Revisión del código					
Compilación					
Pruebas					
Total		6 i	20	100,0	
Defectos eliminados	Plan	Real	Hasta la Fecha	% Hasta la Fecha	Def./Hora
Planificación					
Diseño					
Codificación					
Revisión del código					
Compilación					
Pruebas					
Total		2 t	5 k	25,0	
		2 t	10 k	50,0	
		2 t	5 k	25,0	
		6 t	20	100,0	

1. Anotó los valores reales del Tiempo por Fase (m), del Cuaderno de Registro de Tiempo.
2. Anotó los valores reales de los defectos Introducidos y Eliminados (n), del Cuaderno de Registro de Defectos.
3. Contabilizó 47 LOC Nuevas & Cambiadas en el programa y las anotó bajo la columna Real (o).

4. Con estos datos, calculó los Minutos/LOC reales a partir de los minutos totales, 229 (m), dividido por las 47 (o)LOC Nuevas & Cambiadas, obteniendo $229/47 = 4,87(p)$.
5. Calculó las LOC/Hora reales a partir de los minutos/LOC reales (p) de esta forma: $60/4,87 = 12,32(q)$.
6. Calculó los defectos/KLOC reales a partir del total de defectos reales (n) y de las LOC (o)reales, $1.000*5/47=106,4(r)$.
7. A continuación, calculó los valores Hasta la Fecha para el Tiempo por Fase, sumando los tiempos en las fases para este proyecto a los tiempos Hasta la Fecha para los proyectos anteriores (s).
8. También calculó los valores para los defectos introducidos y eliminados Hasta la Fecha, sumando los defectos reales en cada fase a los valores de los defectos Hasta la Fecha de proyectos anteriores (t).
9. Los valores de % Hasta la Fecha se calcularon también, como se ve en (u).
10. A continuación, calculó el valor Hasta la Fecha para los Minutos/LOC, dividiendo el tiempo Total Hasta la Fecha, 1.479 minutos (s) por el valor de LOC Total Nuevas & Cambiadas, 258(w). Obteniendo $1.479/258 = 5,73(x)$.
11. Utilizando el valor 5,73(x), calculó el valor Hasta la Fecha de LOC/Hora, $60/5,73=10,47(y)$.
12. Finalmente, calculó los defectos/KLOC Hasta la Fecha, multiplicando 1.000 por los defectos totales introducidos Hasta la Fecha (t) y dividiendo el resultado por las LOC Totales Nuevas & Cambiadas Hasta la Fecha (w), obteniéndose $1.000*25/258=96,90(z)$.

Obsérvese que las columnas Hasta la Fecha y % Hasta la Fecha muestran el tiempo total y la distribución de defectos. Con más experiencia, puedes encontrar que tu productividad y el grado de defectos cambia tan significativamente que los datos de los programas iniciales no sirven de guía útil para las planificaciones actuales. En esta situación, puedes desear recalcular todos los valores de Hasta la Fecha y % Hasta la Fecha sin los valores de los primeros programas.

RESUMEN

En este capítulo, utilizas los datos de los defectos para estimar el número de defectos que introducirás y eliminarás en cada fase de un nuevo programa. Conforme aprendas a hacer las previsiones de los defectos,

también aprenderás a hacer mejores planes de desarrollo y un mejor control del número de defectos que introduces y eliminas.

Para hacer una proyección de defectos, necesitas los datos de los defectos y del tamaño de los programas anteriores, y un tamaño estimado para el nuevo programa. Para calcular el número de defectos esperados, supón que los defectos/KLOC para este programa serán iguales a la media de los programas que hayas desarrollado previamente. También asume que el porcentaje de introducción y eliminación de defectos será similar. Utiliza los valores de los defectos/KLOC y % Hasta la Fecha del número de defectos del programa anterior, para hacer los cálculos de planificación.

Se hace una actualización del Resumen del Plan del Proyecto para incorporar la estimación de defectos. Haz el plan de valores durante la fase de planificación para cada proyecto de codificación. Calcula los valores de defectos/KLOC Planificados, Reales y Hasta la Fecha durante la fase postmortem de cada nuevo programa y utiliza estos datos para planificar el siguiente proyecto.

EJERCICIO 15

Para el siguiente programa que escribas, determina el porcentaje de defectos/KLOC y la estimación de defectos introducidos y eliminados por fase. Anota estos datos en el Resumen del Plan del Proyecto de la Tabla 15.3.

Entrega copias del Cuaderno de Registro de Tiempo y de las hojas del Resumen Semanal de Actividades que no hayas entregado previamente. También, entrega copias completas del Resumen del Plan del Proyecto, del Cuaderno de Registro de Defectos y de la Lista de Comprobación para la Revisión de Código para cada programa que desarrolles. Incluye los valores planificados y reales de los tiempos de desarrollo y de los defectos introducidos y eliminados.

REFERENCIA

[Humphrey 95] Humphrey, W.S. *A Discipline for Software Engineering*. Reading, MA: Addison-Wesley, 1995.

CAPITULO 16

La economía de eliminar defectos

Este capítulo trata sobre el ahorro que supone eliminar los defectos. Eliminar los defectos es una cuestión económica, ya que muchas decisiones sobre la eliminación de defectos tienen que ver con el equilibrio entre coste, planificación y calidad. Aunque la calidad del producto debe tener siempre una alta prioridad, muchos proyectos están limitados por la planificación y los costes. Así, el coste relativo de la eliminación de defectos durante el desarrollo, el impacto de los defectos que quedan sobre el cliente y el coste resultante del soporte al cliente, son cuestiones importantes en el desarrollo del software. En el ejercicio de este capítulo, analizarás las tasa de defectos introducidos y eliminados en los programas que has desarrollado hasta ahora en este curso.

16.1

LA NECESIDAD DEL TRABAJO DE CALIDAD

A pesar de los distintos avances tecnológicos que se han producido en el campo de los computadores, el trabajo de los ingenieros software no ha experimentado cambios en los últimos 30 años. Los ingenieros software siempre dividen un gran programa a realizar en pequeñas partes. Y generalmente, trabajan en el desarrollo de una o más de estas partes. En este proceso utilizan el 30-40% o más de su tiempo intentando hacer que es-

tos programas funcionen con pruebas básicas. Una vez que el programa funciona, los programadores, de una forma progresiva los prueban e integran con otros programas hasta formar un gran sistema. El proceso de integración y pruebas esta orientado a encontrar y eliminar los defectos. Cuando el producto final se entrega, a menudo su calidad es tan escasa que los ingenieros deben dedicar muchos meses a corregir los defectos que encuentran los clientes. Esto era una práctica común hace 30 años y lo es hoy en día también.

Hay mejores formas de desarrollar software. El problema básico es que cuando los ingenieros software escriben primero sus programas, normalmente pasan muy rápidamente por el diseño y la codificación, para obtener sus programas en las fases de compilación y pruebas. Un principio del trabajo de calidad, es obtener el producto correcto a la primera. Si se aprende lo anterior, no será necesario eliminar estos defectos en las fases de compilación y pruebas, y tanto tú como tu organización no necesitaréis dedicar tiempo a encontrar y corregir defectos.

Un ejemplo de algunos ingenieros de Motorola muestra la importancia de lo anteriormente dicho. **Su** software se utilizaba para poner en funcionamiento una fábrica de productos avanzados. Si el producto o el proceso de producción lo cambiaban, los ingenieros tenían que cambiar el software. Para encontrar los defectos en cada nueva versión del software, la línea de producción tenía que parar, para hacer las pruebas. Después de cada prueba, la línea de producción era interrumpida periódicamente por los defectos del software que habían quedado ocultos. En la primera actualización de software, después de que los ingenieros hicieran la formación del PSP, hubo solamente un defecto en las pruebas y ninguno en los siguientes meses de producción. Esto no solo ahorró tiempo de pruebas, sino también redujo las interrupciones durante la fabricación. Como resultado, se obtuvo que la planta de fabricación podía producir más productos, y la compañía podía obtener más ingresos.

16.2

EL PROBLEMA DE ELIMINAR DEFECTOS

Eliminar los defectos es costoso porque los defectos son difíciles de encontrar y corregir. Conforme haces sistemas software más grandes y complejos, este problema se agravará. El tamaño y complejidad de los sistemas software aumenta unas diez veces cada diez años. **Así**, las primeras impresoras láser necesitaban unos programas de 20.000 LOC para gestionarlas, la última versión utiliza entorno al millón de LOC. Hace diez años, los automóviles no tenían software; los coches modernos contienen muchas miles de líneas de software. Puesto que el tamaño y la complejidad del software probablemente continuarán creciendo, la eliminación de defectos continuará siendo más costosa y consumirá más tiempo.

Para entender y controlar los costes de los defectos, es fundamental medir la efectividad de la eliminación de defectos. Una de dichas medidas, es el número de defectos eliminados en una hora. Otra medida útil es el rendimiento en la eliminación de defectos. El rendimiento mide la tasa de los defectos encontrados con un método de eliminación. Así, si un producto contenía 100 defectos antes de hacer las pruebas, y se encontraron 45 defectos durante la realización de las pruebas, el rendimiento de dichas pruebas sería del 45%. Cuando sabes el rendimiento y la tasa de eliminación para cada método de eliminación de defectos, puedes decidir mejor cómo encontrar y corregir los defectos. El rendimiento y las medidas de eliminación de defectos se discuten en este capítulo, y en los siguientes capítulos se muestra cómo se utilizan estas medidas.

16.3

EL TIEMPO DE ELIMINAR DEFECTOS

Con programas de unas pocas docenas a cientos de LOC, probablemente introducirás pocos defectos. Aunque encontrarlos y corregirlos será una molestia, llevará poco tiempo. Con grandes programas, sin embargo, el tiempo necesario es mucho mayor. Por ejemplo, 250 ingenieros del sistema NT de Microsoft dedicaron un año completo a encontrar y corregir 30.000 defectos [Zachary]. Esto equivale a una media de **16** horas por defecto.

En grandes sistemas, cuando el ingeniero que está desarrollando un programa no elimina todos los defectos introducidos, otro deberá encontrarlos y corregirlos. Cuantos más defectos haya y se corrijan posteriormente, mayores son los costes del trabajo de reparación. Con grandes productos, estos costes pueden crecer de forma alarmante. Como se indicó en el Capítulo 13, un ingeniero puede encontrar y corregir un defecto en unos pocos minutos durante la revisión personal del código. En las pruebas de unidad, los costes son varias veces superiores, oscilando el tiempo de 15 a 30 minutos. Los defectos que los ingenieros cometen en su trabajo personal, sin embargo, deben detectarse en pruebas posteriores y cuando el usuario utiliza el producto software. Entonces, lleva muchas horas encontrar y corregir cada defecto.

16.4

EXPERIENCIA EN LA INTRODUCCIÓN Y ELIMINACIÓN DE DEFECTOS

Los estudiantes y los ingenieros, normalmente introducen de 1 a 3 defectos por hora durante el diseño y de 5 a 8 defectos cuando escriben el programa. Solamente eliminan de 2 a 4 defectos por hora en las pruebas, pero

encuentran de 6 a 12 defectos por hora durante la revisión personal del código. La Figura 16.1 muestra la tasa de introducción de defectos para una clase de 14 ingenieros y la Figura 16.2 muestra la tasa de eliminación de defectos. Antes de comenzar a escribir el programa 1, véase la parte izquierda de la gráfica, los estudiantes no hacían revisiones de código o seguimiento de las tasas de introducción y eliminación de defectos. Del programa 1 al programa 10, estos ingenieros aprendieron a calcular las tasas de introducción y eliminación de defectos, y fueron adquiriendo habilidad en los métodos de eliminación de defectos del PSP.

Hay varios puntos interesantes en los datos de las Figuras 16.1 y 16.2. Primero, las tasas mostradas en dichas figuras, son los valores medios para los 14 ingenieros. Estos valores parecen razonablemente normales. La tasas individuales de los ingenieros son más variables. Sus tasas personales, probablemente tendrán grandes variaciones de un programa al siguiente. Pero utilizando el PSP para controlar su trabajo, mejorarán inmediatamente.

Los defectos eliminados por hora en las pruebas oscilan de forma consistente entre 2 y 4. Tasas similares se encontraron con estudiantes inexpertos y con ingenieros muy experimentados.

La revisión de código es una habilidad. Con la práctica, muchos ingenieros pueden alcanzar tasas de eliminación de defectos de entre 8 a 10 defectos por hora. La clave está en utilizar los métodos descritos en los Capítulos 13 y 14.

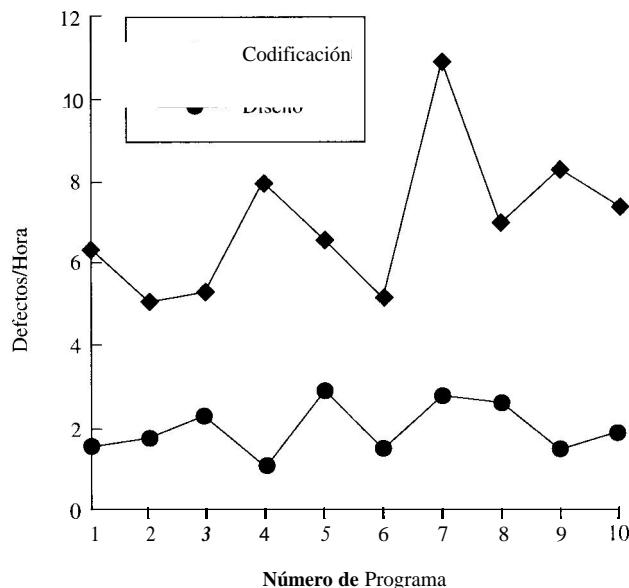


Figura 16.1 Tasas de introducción de defectos

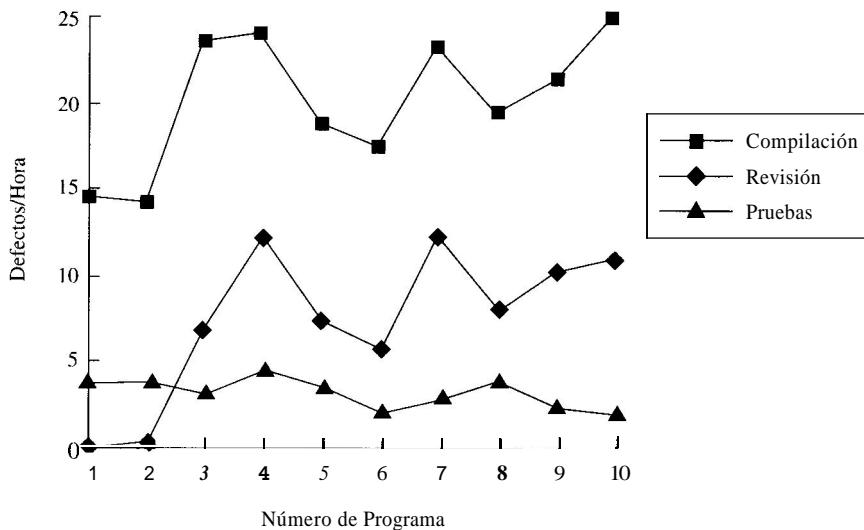


Figura 16.2 Tasas de eliminación de defectos.

La gestión de defectos es un poco como trepar por una escalera que baja. Cuando diseñas y codificas, introduces defectos. Conforme encuentras y eliminas estos defectos, dedicarás más tiempo a rediseñar y a codificar. Durante este tiempo, también introducirás más defectos. Para ganar esta carrera, necesitas reducir la tasa de introducción de defectos y aumentar la tasa de eliminación. Para hacer esto, debes calcular y controlar las tasas de introducción y eliminación de defectos.

Los datos reunidos con el PSP son todos necesarios para calcular las tasas de introducción y eliminación de defectos personales.

16.5 AHORRO EN LA ELIMINACIÓN DE DEFECTOS

Entender las tasas de eliminación de defectos es importante para grandes proyectos y puede ser útil para los ingenieros software. Cuando estás desarrollando un programa de tamaño moderado, y asumes que tus datos son similares a los de otros ingenieros, introducirás unos 100 defectos/KLOC, encontrarás alrededor de 50 defectos/KLOC en la compilación, y otros 40 defectos/KLOC en las pruebas de unidad.

Como se muestra en la Tabla 16.1, esto significa que las 500 LOC de programa tendrían un total de 50 defectos. De estos, 25 se encontrarían en la compilación, y con suerte otros 20 en las pruebas de unidad. Si no usas la revisión de código, necesitarías alrededor de 2 horas para conseguir un programa libre de errores de compilación y encontrar esos 25 defectos. Las pruebas de unidad necesitarían unas 10 horas para encontrar los 20

Tabla 16.1 Ejemplos de introducción y eliminación de defectos.

	500 LOC		10.000 LOC	
	Sin PSP	Con PSP	Sin PSP	Con PSP
Defectos				
Defectos Totales	50	25	1.000	500
Encontrados en la revisión de código	0	15	0	300
Defectos que quedan	50	10	1.000	200
Encontrados en la compilación	25	5	500	100
Encontrados en las pruebas de unidad	20	4	400	80
Defectos que queda	5	1	100	20
Tiempo (horas)				
Tiempo de revisión de código	0	2,5	0	50
Tiempo de compilación	2	0,5	40	10
Tiempo de pruebas de unidad	10	2	200	40
Tiempo personal de eliminación de defectos	12	5	240	100
Tiempo de integración y pruebas del sistema			1.000	200
Tiempo total de eliminación de defectos	12	5	1.240	300

defectos esperados. Es decir, un total de 12 horas dedicadas a la eliminación de defectos.

Después de aprender del PSP, solamente introducirías alrededor de 50 defectos por KLOC (25 defectos). Con las revisiones de código, encontrarías entre el 60 y el 70% de estos defectos antes de la primera compilación. De esta forma, la compilación supone unos 30 minutos y quedan de 4 a 8 defectos que se han de encontrar en las pruebas de unidad. Las pruebas suponen alrededor de unas 2 horas. Suponiendo que dedicas unas 2,5 horas a la revisión de código, el tiempo total para la eliminación de los defectos sería de unas 5 horas. Esto supone un ahorro de 7 horas.

Aunque no pueda parecer mucho esfuerzo, reducir el tiempo de eliminación de defectos en 7 horas para un trabajo de varios días, considera cómo se multiplican estos números. Si en vez de un programa de 500 LOC, desarrollásemos uno de 10.000 LOC, comenzarías con 1000 defectos. Con un entrenamiento en el PSP, serían solo 500 defectos. Si dedicas 50 horas a la revisión de código, reducirías el tiempo de compilación de 40 a 10 horas y el tiempo de pruebas de unidad de 200 a 40 horas. Las 140 horas que ahorras son equivalentes a varias semanas de trabajo.

De mayor significado, es el coste de los defectos que tú no encuentras en tu proceso. En el caso de las 500 LOC, los defectos eran unos pocos. Con el Componente más grande, alrededor de unos 100 defectos ocultos se encontrarían en las pruebas y muchos más por el usuario. Encontrar esos 100 defectos llevaría normalmente unas 1000 horas. Con el proceso del PSP, solamente se dejarían unos 20 defectos que se encontrarían más tarde en las pruebas, reduciendo el tiempo de las pruebas en unas 800 horas.

Para ver lo que esto puede significar en la práctica, supón que tú y otros cuatro ingenieros desarrolláis un producto software de 50.000 LOC. Puedes planificar que cada uno desarrolla un componente de 10.000 LOC y posteriormente se integra y prueba el sistema completo. Basándonos en los datos típicos de un ingeniero, tú y tus compañeros probablemente introduciréis unos 100 defectos/KLOC en el programa. Esto significa que habrá que encontrar y corregir 5000 defectos. Utilizando tasas normales, unos 2500 de estos defectos se encontrarán durante la fase de compilación y otros 2000 en las pruebas de unidad personales. Esto nos deja unos 500 defectos a detectar en la integración y pruebas del sistema. Asumiendo que tu producto era más sencillo que el NT de Microsoft, puedes encontrar estos defectos a un coste medio de solo unas 10 horas. Eliminar estos defectos le supondría a tu equipo de cinco personas unas 5000 horas. Si el equipo no hace más de 40 horas semanales, esto sería un trabajo de 6 meses. Si hubieses revisado personalmente todo tu trabajo y dejás que el equipo inspeccione tus programas, podrías ahorrarte al menos cinco meses de pruebas. Para un proyecto de dos años, esta diferencia en el tiempo de pruebas podría ser la diferencia entre entregar a tiempo el producto o sufrir un serio retraso.

16.6

EL CÁLCULO DE LOS DEFECTOS/HORA EN EL RESUMEN DEL PLAN DEL PROYECTO DEL PSP

La Tabla 16.2 muestra cómo se calculan las tasas de eliminación de defectos Hasta la Fecha en la tabla Resumen del Plan del Proyecto. Para cualquier fase, comienza con el número de defectos introducidos Hasta la Fecha en esa fase y el número de minutos dedicados Hasta la Fecha en dicha fase. A continuación calcula los defectos/hora introducidos Hasta la Fecha de esta forma: $60 * (\text{defectos introducidos Hasta la Fecha en una fase}) / (\text{minutos dedicados Hasta la Fecha en esa fase})$. Los defectos eliminados por hora se calculan así: $60 * (\text{defectos eliminados Hasta la Fecha en una fase}) / (\text{minutos dedicados Hasta la Fecha en esa fase})$. En el ejemplo de la Tabla 16.2, estos cálculos se muestran en la columna de la derecha de las secciones de Defectos introducidos y Defectos eliminados. Estos cálculos se hacen como se muestra a continuación.

Tabla 16.2 Resumen del plan del proyecto del PSP.

Estudiante	Estudiante X	Fecha	28/11/96
Programa		Programa#	14
Profesor	Sr. Z	Lenguaje	Ada
Resumen	Plan	Real	Hasta la Fecha
Minutos/LOC	5,73	4,65	5,48
LOC/Hora	10,47	12,90	10,95
Defectos/KLOC	96,90	77,9	92,53
Rendimiento	33,3	80,0	40,0
V/F			
Tamaño Programa (LOC):			
Total Nuevo & Cambiado	67	77	335
Tamaño Máximo	85		
Tamaño Mínimo	49		
Tiempo por Fase (min.)	Plan	Real	Hasta la Fecha
Planimación	23	32	120
Diseño	39	44	195
Codificación	166	155	792
Revisión del código	29	34	145
Compilación	24	8	100
Pruebas	62	39	279
Postmorten	4	46	206
Total	384	358	1037
Tiempo Máximo	407		
Tiempo Mínimo	281		
Defectos introducidos	Plan	Actual	Hasta la Fecha
Planimación			
Diseño	1	1	5
Codificación	5	4	25
Revisión del código			
Compilación			1
Pruebas			3,2
Total	6	5	3
Defectos eliminados	Plan	Actual	Hasta la Fecha
Planimación			
Diseño			
Codificación			
Revisión del código	7	4	12
Compilación	3	1	13
Pruebas	1	1	6
Total	6	5	3

Para los defectos introducidos por hora:

Defectos introducidos Hasta la Fecha en la fase de diseño

= 60*(defectos introducidos Hasta la Fecha en el diseño)/(minutos dedicados Hasta la Fecha al diseño)

$$= 60*5/195$$

$$= 1.54$$

Defectos introducidos Hasta la Fecha en la fase de codificación

$$\begin{aligned} &= 60 * (\text{defectos introducidos Hasta la Fecha en la codificación}) / (\text{minutos dedicados Hasta la Fecha a la codificación}) \\ &= 60 * 25 / 792 \\ &= 1,89 \end{aligned}$$

Aunque podrías calcular las tasas de introducción de defectos para las otras fases, estas dos fases que hemos visto son las **más** importantes. Para la tasa de eliminación de defectos tenemos:

Defectos eliminados Hasta la Fecha en la fase de revisión de código

$$\begin{aligned} &= 60 * (\text{defectos eliminados hasta la Fecha en la revisión de código}) / (\text{minutos dedicados Hasta la Fecha a la revisión de código}) \\ &= 60 * 12 / 145 \\ &= 4,97 \end{aligned}$$

Defectos eliminados Hasta la Fecha en la fase de compilación

$$\begin{aligned} &= 60 * (\text{defectos eliminados hasta la Fecha en la compilación}) / (\text{minutos dedicados Hasta la Fecha a la compilación}) \\ &= 60 * 13 / 100 \\ &= 7,80 \end{aligned}$$

Defectos eliminados Hasta la Fecha en la fase **de** pruebas

$$\begin{aligned} &= 60 * (\text{defectos eliminados Hasta la Fecha en las pruebas}) / (\text{minutos dedicados Hasta la Fecha a las pruebas}) \\ &= 60 * 6 / 279 \\ &= 1,29 \end{aligned}$$

Aunque estos cálculos se realizan solamente para los valores de Hasta la Fecha, podrías calcular, si lo deseas, los defectos por hora para los datos planificados y reales. Observa que estos valores cambiarán conforme cambies tu proceso. También, el número de defectos es un poco inferior al que debería ser, puesto que no se registraron los datos de defectos de los primeros programas. Las instrucciones para actualizar el Resumen del Plan del Proyecto se muestran en la Tabla 16.3.

16.7

EL CÁLCULO DEL RENDIMIENTO EN EL RESUMEN DEL PLAN DEL PROYECTO

Cuando eliminamos defectos en una fase, nos gustaría saber cuántos se encontraron y cuántos quedaron ocultos. Aunque no hay forma de saber esto durante el desarrollo, los datos de procesos históricos pueden ayudar a tener una ligera idea. La forma de hacer esto es calcular y controlar el rendimiento. Para el PSP, definimos el rendimiento del proceso como la tasa de defectos encontrados antes de la primera compilación y las pruebas.

Tabla 16.3 Instrucciones del resumen del plan del proyecto del PSP

Propósito	Esta tabla trata los datos estimados y reales de los proyectos de una forma cómoda y fácilmente recuperable.
Cabecera	Introduce los siguientes datos: <ul style="list-style-type: none"> • Tu nombre y fecha de hoy. • Nombre y número de programa. • Nombre del profesor. • El lenguaje que utilizarás para escribir el programa.
Minutos/LOC	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> • Escribe los Minutos/LOC planificados para este proyecto. Utiliza la velocidad Hasta la Fecha de un programa reciente del Cuaderno de Registro de Trabajos o del Resumen del Plan del Proyecto. Después del desarrollo: <ul style="list-style-type: none"> • Divide el tiempo total de desarrollo por el tamaño real del programa para obtener los Minutos/LOC reales y los Minutos/LOC Hasta la Fecha. • Por ejemplo, si el proyecto se hizo en 196 minutos e hiciste 29 LOC, los Minutos/LOC serían $196/29 = 6,76$.
LOC/Hora	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> • Calcula las LOC por hora planificada para este programa dividiendo 60 por los Minutos/LOC de la casilla de Plan. Después del desarrollo: <ul style="list-style-type: none"> • Para LOC/Hora Real y Hasta la Fecha divide 60 por Minutos/LOC Reales y Hasta la Fecha. • Para los 6,76 Minutos/LOC Reales, tenemos $60/6,76 = 8,88$ LOC/Hora Reales.
Defectos/KLOC	Antes de desarrollar: <ul style="list-style-type: none"> • Encuentra los defectos/KLOC Hasta la Fecha del programa más reciente. • Utiliza dicho valor como los defectos/KLOC planificados. Después del desarrollo: <ul style="list-style-type: none"> • Calcula los valores de defectos/KLOC reales y Hasta la Fecha para este programa. • Para el valor real, multiplica el total de defectos reales por 1000 y divídalo por las LOC Reales Total Nuevas & Cambiadas. • Para el valor Hasta la Fecha haz unos cálculos similares. • Con 17 defectos Hasta la fecha y 153 LOC Total Nuevas & Cambiadas, los defectos/KLOC Hasta la Fecha = $1000*17/153=111,11$
Rendimiento	Calcula el rendimiento planificado, real y Hasta la Fecha. Rendimiento = $100 * (\text{defectos eliminados antes de compilar}) / (\text{defectos introducidos antes de compilar})$, así, con 5 defectos introducidos y 4 localizados, el rendimiento = $100 * 4/5 = 80,0\%$

(Continúa)

Tabla 16.3 Instrucciones del resumen del plan del proyecto del PSP.
(Continuación)

Tamaño Programa (LOC)	Antes de iniciar el desarrollo. • Escribe bajo la columna plan, el valor estimado de LOC Total Nuevas & Cambiadas, Máximo y Mínimo. Después del desarrollo: • Cuenta y escribe las LOC Nuevas & Cambiadas Reales. • Para la columna Hasta la Fecha, añade LOC Reales Nuevas & Cambiadas a las LOC Hasta la Fecha Nuevas & Cambiadas de programas anteriores.
Tiempo por Fase Plan	Para el tiempo total de desarrollo , multiplica el valor de las LOC Total Nueva & Cambiada por los Minutos/LOC. Para el tiempo Máximo, multiplica el tamaño Máximo por los Minutos/LOC. Para el tiempo Mínimo, multiplica el tamaño Mínimo por los Minutos/LOC. Del Resumen del Plan del Proyecto de un programa reciente, busca los valores de % Hasta la Fecha para cada fase. Utilizando el % Hasta la Fecha de programas anteriores calcula el tiempo planificado para cada fase.
Real	Una vez acabado el trabajo, anota el tiempo real en minutos que has gastado en cada fase del desarrollo. Obtén estos datos del Cuaderno de Registro de Tiempos.
Hasta la Fecha	Para cada fase, escribe la suma del tiempo real y el tiempo Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, escribe 100 multiplicado por el tiempo Hasta la Fecha y lo divides por el total del tiempo Hasta la Fecha.
Defectos Introducidos Plan	Antes del desarrollo, estima el número total de defectos a introducir en el programa. El valor es: defectos/KLOC planificados multiplicado por las LOC Total Nuevas & Cambiadas planificadas para este programa y dividido por 1000. Por ejemplo, con un valor de defectos/KLOC planificado de 75,9 y un valor de LOC Nuevas & Cambiadas planificadas de 75, el total de Defectos Planificados = $75,9 \times 75 / 1000 = 5,69$, se redondea a 6. Antes de desarrollar, estima los defectos introducidos por fase utilizando el total de defectos estimados y el % Hasta la Fecha de defectos introducidos de programas anteriores.
Real	Después del desarrollo, localiza y anota el número real de defectos introducidos en cada fase.
Hasta la Fecha	Para cada fase, escribe la suma de los defectos reales y los defectos Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, multiplica por 100 los defectos Hasta la Fecha para esa fase y divídelos por el total de defectos Hasta la Fecha.

(Continúa)

Tabla 16.3 Instrucciones del resumen del plan del proyecto del PSP.
(Continuación)

Defectos/hora	<i>Calcula los defectos introducidos por hora para la fase de diseño y codificación. Para la fase de diseño, por ejemplo, se multiplica 60 por los defectos de diseño Hasta la Fecha y se divide por el tiempo de diseño Hasta la Fecha = $60*5/195=1,54$ defectos/hora.</i>
Defectos eliminados Plan	En la fila de total, escribe el total de defectos estimados. Utiliza los valores de % Hasta la Fecha de los programas más recientes, calcula los defectos eliminados planificados por cada fase.
Real	Después del desarrollo, localiza y anota el número real de defectos eliminados en cada fase.
Hasta la Fecha	Para cada fase, escribe la suma de los defectos reales y los defectos Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, multiplica por 100 los defectos Hasta la Fecha para esa fase y divídelos por el total de defectos Hasta la Fecha.
Defectos/hora	<i>Calcula los defectos eliminados por hora para la revisión de código, compilación y pruebas. Para las pruebas, por ejemplo, multiplica 60 por los defectos de las pruebas Hasta la Fecha y divídalo por el tiempo de pruebas Hasta la Fecha = $60*6/279=1,29$ defectos/hora.</i>

Los cálculos para el rendimiento Planificado, Real y Hasta la Fecha se muestran en la sección resumen en la parte superior de la tabla Resumen del Plan del Proyecto (véase Tabla 16.2). Bajo la columna Planificado, por ejemplo, los valores de rendimiento del proceso se calculan de la siguiente manera:

Los defectos introducidos planificados antes de compilar son $1+5=6$.

Los defectos eliminados planificados antes de compilar son 2.

El valor planificado de rendimiento del proceso es:

Rendimiento_{planificado}

$$\begin{aligned}
 &= 100 * (\text{Plan de defectos eliminados antes de compilar}) / (\text{Plan de defectos introducidos antes de compilar}) \\
 &= 100 * 2 / (1 + 5) \\
 &= 100 * 2 / 6 \\
 &= 33,3\%
 \end{aligned}$$

El valor del rendimiento real se calcula así:

Rendimiento_{Real}

$$\begin{aligned}
 &= 100 * (\text{Defectos eliminados reales antes de compilar}) / (\text{Defectos introducidos reales antes de compilar}) \\
 &= 100 * 4 / (1 + 4)
 \end{aligned}$$

$$\begin{aligned} &= 100 * 4 / 5 \\ &= 80,0\% \end{aligned}$$

y el valor Hasta la fecha:

Rendimiento_{Hasta la Fecha}

$$\begin{aligned} &= 100 * (\text{Defectos eliminados Hasta la Fecha antes de compilar}) / (\text{Defectos introducidos Hasta la Fecha antes de compilar}) \\ &= 100 * 12 / (5 + 25) \\ &= 100 * 12 / 30 \\ &= 40,0\% \end{aligned}$$

Obsérvese que en la Tabla 16.2, se introdujo un defecto durante la compilación. Puesto que el cálculo del rendimiento del proceso concierne a la tasa de defectos eliminados antes de compilar, deberías considerar solamente los defectos introducidos y eliminados antes de la fase de compilación. Cualquier defecto introducido en la fase de compilación o pruebas no estaba en el programa durante la revisión del código y no se incluye en los cálculos de rendimiento del proceso.

16.8

LA MEJORA DE LAS TASAS DE ELIMINACIÓN DE DEFECTOS

Puedes mejorar rápidamente la tasa de eliminación de defectos haciendo revisiones de código. Una vez que has obtenido estos beneficios iniciales, las mejoras posteriores serán más difíciles de conseguir. Puesto que los cambios a los que se deben enfrentar los ingenieros software aumentan cada año, no puedes permitirte dejar de mejorar. Algunas sugerencias de cómo mejorar tu tasa de eliminación de defectos son:

- Fíjate primero en el rendimiento. Recuerda, que el objetivo es eliminar todos los defectos. Tu primer objetivo debe ser alcanzar de forma consistente un rendimiento del 70% o superior.
- Haz revisión del código antes de la primera compilación. Como se indicó en el Capítulo 13, para alcanzar el máximo rendimiento posible, utiliza el compilador para comprobar la calidad de tu revisión de código.
- Una vez que has conseguido de forma consistente un rendimiento adecuado, utiliza los métodos descritos en los Capítulos 13 y 14 para mejorar las tasas de revisión. Controla en la lista de comprobación dónde se encuentran y se pasan por alto defectos y haz ajustes de forma periódica. Si en algún paso no encuentras o pasas por alto muchos defectos, considera eliminarlo. Cuando se pasan por alto defectos, considera añadir un paso en la lista de comprobación que se centre específicamente en este tipo de defectos.

Recuerda que es una insensatez hacer lo mismo una y otra vez y esperar un resultado diferente [Brown]. Si no cambias la lista de comprobación, pasarás por alto los mismos defectos.

Continúa reuniendo datos de defectos y calculando el rendimiento y las tasas de introducción y de eliminación de defectos. Controla estos datos y experimenta con distintas aproximaciones para ver cuáles te ayudarán a mejorar.

Un objetivo de este libro es que seas tan consciente de los costes y consecuencias de los defectos que personalmente llegues a estar comprometido a producir programas de alta calidad. Esto hará que se ahorre tiempo. Tal y como se observa en la Figura 16.3, los estudiantes que controlaron y gestionaron su rendimiento y sus tasas de eliminación de defectos, dedicaron menos tiempo a la compilación y a las pruebas. Al principio del curso, en los programas 1 y 2, los 14 estudiantes dedicaron alrededor de un 30% de su tiempo a la compilación y a las pruebas. Pero al finalizar el curso, cuando utilizaron consistentemente el PSP, el tiempo medio de compilación y pruebas fue un 10% del tiempo de desarrollo.

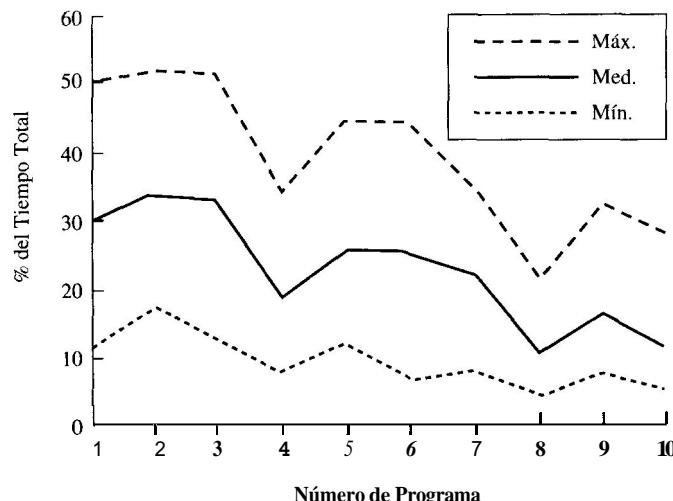


Figura 16.3 Tiempo de compilación y pruebas.

16.9

LA REDUCCIÓN DE LAS TASAS DE INTRODUCCIÓN DE DEFECTOS

Las tasas de introducción de defectos son difíciles de reducir puesto que se introducen defectos en cada parte del proceso. Por ello, necesitas considerar cada tarea del software. Algunas aproximaciones para reducir las tasas de introducción de defectos son las siguientes:

1. *Registrar todos tus defectos.* Estando enterado de los defectos, trabajarás más cuidadosamente y reducirás el número de defectos que introduces.
- 2 *Hacer mejores diseños.* Haciendo diseños más completos y mejor documentados, puedes mejorar la calidad de tus programas de dos formas. Primero, evitarás aquellos defectos que podrían resultar de un diseño confuso o incompleto. Segundo, un diseño más completo te ahorrará tiempo de codificación. Puesto que las tasas de introducción de defectos son más bajas durante la fase de diseño que durante la fase de codificación, esto también reducirá los defectos. Esta cuestión se discute en el Capítulo 17.
3. *Utilizar los mejores métodos.* Puesto que los defectos pueden introducirse en cualquier fase, las mejoras en la forma de desarrollar los requisitos, especificaciones, diseños, casos de pruebas y código fuente ayudarán a reducir los defectos. Muchos cursos de ingeniería del software enseñan métodos útiles para estas actividades. Utilizando el PSP y midiendo tu trabajo con estos métodos, puedes ver cómo funcionan para ti.
4. *Utilizar herramientas mejores.* Si una herramienta ahorra tiempo, probablemente reducirá el número de defectos que se introducen. Cada año se desarrollan nuevas herramientas de software y los datos del PSP permitirán medirlas y evaluarlas. Puedes ver cuál te ayuda y cuánto.

RESUMEN

Los defectos son una de las principales causas de los problemas del software. Su corrección supone un coste económico y un retraso en la planificación del proyecto. Puesto que los tiempos dedicados a corregir defectos son impredecibles, los defectos también reducen la precisión de la planificación. La gestión de defectos puede, de esta forma, verse como una cuestión económica. Este capítulo explica las consecuencias económicas de los defectos e introduce medidas para ayudar a gestionar los defectos en los programas que hagas.

Hasta los ingenieros experimentados introducen muchos defectos. El coste de encontrarlos aumenta unas 10 veces en cada fase posterior a la compilación. Las tasas típicas de eliminación de defectos muestran que las pruebas son una forma ineficiente de eliminar defectos. En las pruebas de unidad, los ingenieros solamente encuentran de 2 a 4 defectos por hora. En la revisión de código, encuentran y corrigen de 6 a 12 defectos por hora. Los compiladores son muy eficientes para encontrar defectos sintácticos, pero no encontrarán todos los defectos sintácticos de tu programa.

El tiempo para encontrar y corregir cada defecto pasado por alto en un proceso personal llevará varias horas entre la integración, las pruebas del sistema y la utilización del producto.

Para mejorar tu efectividad en encontrar y corregir defectos, se han introducido dos nuevas medidas: los Defectos/Hora que miden la eficiencia de la fase para introducir y eliminar los defectos. En el PSP, el rendimiento del proceso mide la tasa de defectos eliminados antes de la primera compilación. Para ver dónde y cómo mejoras tu proceso, calcula las tasas de introducción y eliminación de defectos y el rendimiento del proceso.

EJERCICIO 16

Para cada uno de los programas que has escrito en este curso, encuentra la tasa de defectos introducidos Hasta la Fecha en las fases de diseño y codificación y la tasa de defectos eliminados Hasta la Fecha para la revisión de código, compilación y pruebas. También calcula el rendimiento del proceso para cada uno de estos programas. Calcula y entrega estos valores junto con un breve resumen que describa las implicaciones de estos datos para ti.

Entrega copias del Cuaderno de Registro de Tiempos y de las hojas del Resumen Semanal de Actividades que no hayas entregado previamente. También, entrega copias completas del Resumen del Plan del Proyecto, del Cuaderno de Registro de Defectos y de la Lista de Comprobación para la Revisión de Código para cada programa que desarrolles. Incluye los valores Planificados, Reales y Hasta la Fecha de los tiempos de desarrollo, de los defectos introducidos, de los defectos eliminados y del rendimiento.

REFERENCIAS

[Brown] Brown, Rita Mae. Comunicación privada.

[Humphrey 89] Humphrey, W.S. *Managing the Software Process*. Reading, MA: Addison-Wesley, 1989.

[Zachary] Zachary, G. Pascal. *Showstopper!* New York: The Free Press, 1994.

CAPÍTULO 17

Defectos de diseño

Este capítulo trata sobre los defectos de diseño, el proceso de diseño y las representaciones del diseño. También trata los métodos para reducir el número de defectos de diseño que introduces. La cuestión principal en la gestión de los defectos de diseño es comprender cuándo y cómo los introduces. Como ejercicio, analizarás los defectos de diseño en tus programas y determinarás cuándo y por qué se introdujeron.

17.1

LA NATURALEZA DE LOS DEFECTOS DE DISEÑO

Muchos de los defectos que encuentras en las pruebas, probablemente fueron introducidos durante la fase de codificación. Esto implica que el principal problema del defecto está relacionado con simples errores de codificación. De hecho, de los defectos que encontraron en las pruebas los 14 estudiantes de uno de mis cursos, introdujeron la mitad de defectos en la fase de diseño que durante la fase de codificación. Puesto que el compilador había eliminado muchos de los errores sintácticos, esto pareció un poco sorprendente. Como se muestra en la Tabla 17.1, esta proporción cambió cuando los estudiantes aprendieron el PSP. De los defectos encontrados en las pruebas, los estudiantes comenzaron a introducir una vez y media más defectos en la codificación que los introducidos durante el diseño. Hacia el final del curso, introdujeron un promedio de solamente un 14 % más. Aunque redujeron el número de defectos introducidos en am-

bos casos, la mejora fue algo menor para los defectos introducidos durante el diseño. Obsérvese que los valores de la Tabla 17.1 se refieren a las fases donde los defectos fueron introducidos. Si examinamos los tipos de defectos, la cuestión es diferente. En el **PSP**, los tipos de defectos están ordenados por grado de sofisticación del problema, los tipos del 10 a 20 son los más sencillos y los tipos del 90 a 100 los más complejos. Podemos decir que los tipos del 10 al 40 son defectos sencillos o de codificación, y los tipos del 50 a 100 son defectos más complejos o de diseño.

Tabla 17.1 Defectos de pruebas introducidos en las fases de diseño y programación.

	Diseño Defectos/KLOC	Codificación Defectos/KLOC	Relación de defectos codificación y diseño
Ejercicio 1	8,66	12,99	1,50
Ejercicio 10	5,05	5,77	1,14
% Reducción	41,67%	55,56%	

Si clasificamos estos mismos defectos de pruebas por tipos, obtendremos la Tabla 17.2. Aquí, la mayor parte de los defectos de pruebas son del tipo diseño, pero muchos de ellos se introdujeron durante la codificación. Aunque estos ingenieros habían reducido bastante el número de defectos introducidos, cometieron muchos de sus errores de diseño durante

	Tipo Diseño Defectos/KLOC	Tipo Codificación Defectos/KLOC	% de defectos de Diseño
Ejercicio 1			
Fase de diseño	7,22	1,44	83,33%
Fase de codificación	9,38	3,61	72,22%
Total	16,59	5,05	76,67%
Ejercicio 10			
Fase de diseño	3,61	1,44	71,43%
Fase de codificación	3,61	2,16	62,50%
Total	7,22	3,61	66,67%
Tasa de Reducción			
Fase de diseño	50,00%	0%	
Fase de codificación	61,54%	40,00%	
Total	56,52%	28,57%	

la fase de codificación. Como vimos en el Capítulo 16, los mismos ingenieros introdujeron de 5 a 8 defectos por hora durante la codificación y solamente de 1 a 3 defectos por hora durante el diseño. Esto sugiere que una forma potencialmente efectiva para introducir pocos defectos de diseño sería dejar de hacer el diseño durante la fase de codificación. En este capítulo se trata este problema y cómo abordarlo.

17.2

IDENTIFICACIÓN DE LOS DEFECTOS DE DISEÑO

Podemos pensar que todos los defectos detectados por un compilador deberían clasificarse como defectos de codificación. Esto implica que los defectos no detectados por el compilador son defectos de diseño. Desafortunadamente, este no es el caso. Los mejores compiladores no encuentran todos los defectos sintácticos, y algunos errores de diseño también podrían producir una sintaxis incorrecta.

La razón de que los compiladores no encuentren todos los defectos sintácticos, se debe a que algunos errores de sintaxis, producen código que tiene validez sintáctica para el compilador. Por ejemplo, al teclear podemos equivocarnos, pero casualmente introducir caracteres que sean válidos para el lenguaje. Esto supone alrededor del 10% de todos los defectos sintácticos. Un ejemplo de esto, sería poner `x = y` cuando tú pretendes escribir `x == y`. El compilador de C++ no detectaría este error. Entonces, en vez de hacer la comprobación lógica de si `x` es igual a `y`, el programa asignaría a `x` el valor de `y`. De forma similar, en Ada o Pascal, podrías tener el mismo problema cuando escribieses erróneamente `x := y`, en vez de poner `x = y`. Aunque estos errores no se consideran errores de diseño. Hacen que la lógica del programa sea incorrecta.

No hay forma sencilla y objetiva de definir los defectos de diseño. Las dos elecciones son:

- Definir todos aquellos defectos introducidos en la fase de diseño como defectos de diseño.
- Definir aquellos tipos de defectos que implican cuestiones de funciones de codificación, lógica, rendimiento y sincronización como defectos de diseño.

En este libro, seguimos la segunda definición.

17.3

¿QUÉ ES DISEÑAR?

Para considerar los defectos de diseño, es importante definir qué entendemos por diseño. Esta cuestión no es fácil. La razón, es que cualquier co-

sa relacionada con la estructura e implementación de un programa está relacionada con el diseño. Esto incluye el flujo del programa, la estructura y naturaleza de los elementos del lenguaje, el uso de los símbolos de separación en el código fuente. Aunque todo lo anterior es parte del diseño, lo es en diferentes grados de detalle.

El problema es que el diseño es una cuestión de perspectiva. Puedes diseñar en detalle o a alto nivel. Cuando desarrollamos programas, hacemos algo de diseño en cada paso. Si se implementan los detalles de una sentencia case o de un bucle, especificamos el diseño detallado. Cuando definimos una estructura de datos o establecemos un módulo de interfaz, estamos trabajando a alto nivel.

La división entre diseño y codificación es un poco arbitraria. Los ingenieros software, normalmente encuentran útil dividir sus actividades de acuerdo al nivel de detalle implicado en su trabajo. Esto les ayuda a centrarse sobre las cuestiones adecuadas en el orden correcto. Por ejemplo, los ingenieros cometen pocos errores y son más eficientes cuando piensan los diseños antes de implementarlos. Esto no significa que hagan todo su trabajo de arriba a abajo. Significa que normalmente comienzan con un concepto de alto nivel antes de entrar profundamente en detalle.

El término que utilizamos para describir una aproximación conceptual de alto nivel, es *abstracción*. Así, cuando hablamos de la abstracción funcional raíz cuadrada, estamos hablando de un procedimiento que calcula la raíz cuadrada. En este nivel, no nos preocupamos de los detalles para calcular la raíz cuadrada. Siguiendo esta estrategia de abstracción, los ingenieros pueden hacer diseños completos de alto nivel antes de profundizar en el embrollo de detalles de cómo se construye cada abstracción o función.

Así, por ejemplo, puedes comenzar a diseñar un programa sencillo, definiendo cuatro funciones abstractas o procedimientos: Entrada, Fichero, Calcular y Salir. Podrías diseñar la rutina del programa principal utilizando estas abstracciones. Aunque tú también podrías tener definidas las variables principales y los ficheros, es posible completar el diseño de alto nivel del programa utilizando estas abstracciones funcionales. El siguiente paso sería diseñar cada una de estas abstracciones. En grandes programas, podrías tener varios niveles de abstracción y podrías construir una librería de abstracciones previamente desarrolladas para utilizarla en futuros programas.

17.4

EL PROCESO DE DISEÑO

Haciendo lo que ellos llaman diseño, los ingenieros experimentados, a menudo, se mueven dinámicamente entre niveles de diseño [Curtis]. La

razón es que ellos están tratando con muchas abstracciones funcionales. Antes de que se sientan cómodos utilizando estas abstracciones en un diseño de alto nivel, generalmente necesitan conocer cómo funcionan dichas abstracciones. Si han utilizado anteriormente dichas funciones, pueden diferir la definición de los detalles. Si no han trabajado nunca con dichas abstracciones, a menudo, se detendrán y completarán su diseño detallado. Pueden escribir y probar un prototipo antes de estar suficientemente satisfechos como para continuar con el diseño de alto nivel. Los diseñadores experimentados hacen esto porque han aprendido que las abstracciones aparentemente sencillas, a menudo, tienen sutiles complicaciones. Así, cuando utilizan una abstracción que no saben cómo construir, frecuentemente encuentran que el trabajo es más difícil de lo esperado. Ocasionalmente, en efecto, cuando los diseños del sistema están basados en abstracciones aparentemente simples, pero mal definidas, se encuentra que la aproximación al diseño global no es factible.

La noción de abstracción es muy útil cuando se trata con funciones conocidas y definidas. Cuando utilizas esta aproximación conceptual para definir lo más arduo del trabajo de diseño, es fácil no tener problemas. Los ingenieros experimentados son reacios a utilizar cualquier abstracción en un diseño de alto nivel, a menos que previamente hayan trabajado con funciones similares o tengan una implementación que sepan cómo funciona.

La distinción entre diseño y codificación es así arbitraria. Esto también significa que la especificación de qué constituye un diseño completo es arbitraria. Por lo tanto, es importante distinguir entre el proceso de diseño y la fase de diseño específico en el PSP. La fase de diseño comienza cuando realizas el producto que llamas diseño. El proceso de diseño describe las tareas necesarias para elaborar este producto. Hasta cierto punto, estamos definiendo diseño como lo que hacemos en la fase de diseño. Esto, probablemente parece tan útil como definir *pensamiento* como lo que la gente hace cuando piensa. A menudo, es útil definir actividades en términos de los productos que obtienes. Posteriormente en este capítulo discutiremos cómo representar el producto del diseño.

17.5

LAS CAUSAS DE LOS DEFECTOS DEL DISEÑO

Los defectos del diseño son causados por varios problemas. El primero es un diseño erróneo. Has trabajado seriamente sobre el problema y tomas una decisión de diseño que era errónea. Quizás no se comprendió una función matemática, se diseñó un bucle bajo en condiciones erróneas, o se ignoró alguna condición del sistema. Puedes haber estado escasamente informado, cansado o inadecuadamente entrenado. A pesar de todas las causas, has tomado conscientemente una decisión de diseño pero incorrecta.

Una segunda causa sería, conociendo el diseño que había que hacer, cometes un simple error. Puedes pensar terminar un bucle bajo ciertas condiciones pero olvidas incluir todos los casos. Esto podría ser un descuido o una tontería. Sabías qué hacer y cómo hacerlo, pero cometiste un error tonto. Estos errores son más comunes cuando tienes prisas o estás cansado.

Una tercera causa de los defectos de diseño es una mala comprensión de lo que se quería. Tanto si has interpretado mal el diseño de alto nivel o no has entendido los requisitos. En cualquier caso, se hace un diseño incorrecto. Realmente el diseño era correcto en el sentido que realizaba la función que pretendías, pero implementaste la función errónea.

Otra causa parecida a la última es cuando se entiende el diseño local y el del sistema, es decir, se comprende los requisitos pero no se entiende el contexto del sistema. Esto es lo que denominó el problema de “interpretación literal”. Por ejemplo, cuando pides a alguien que explique la expresión “derramar las judías”, obtendrás diferentes respuestas dependiendo a quién preguntes. Alguien que entienda el inglés coloquial tal y como se habla en Estados Unidos la interpretará como “decir un secreto”, mientras que alguien que esté aprendiendo inglés en otro país puede decir que significa “tirar la comida”.

Este es un problema en el software, porque muchas decisiones de diseño implican detalles que afectan a los usuarios del sistema. Cuando los diseñadores no han conocido profundamente el contexto del usuario, es probable que interpreten los requisitos de forma literal, sin entender sus implicaciones desde el punto de vista operativo. Aunque dichos sistemas, normalmente cumplen todos los requisitos establecidos, generalmente son poco prácticos y algunas veces es imposible utilizarlos.

Sorprendentemente, los errores de diseño provienen de un exceso de confianza. El ejemplo común sería una o dos líneas corregidas en un gran programa. Puesto que dichos cambios son pequeños, los ingenieros normalmente piensan que son sencillos y no debe llevar mucho tiempo entender realmente el problema o las implicaciones de la corrección. Aunque estos errores podrían ser clasificados en cualquiera de las categorías anteriores, normalmente se ubicarían entre la tercera o cuarta: incomprendición o errores de contexto.

17.6

EL IMPACTO DE LOS DEFECTOS DE DISEÑO

Un ejemplo de problema de diseño común es aquel que me expuso una directora de desarrollo en IBM. Ella había oído por casualidad a dos ingenieros debatir sobre cómo debería ser implementada una función de un programa. Aunque ninguno de los dos sabía con exactitud qué era lo que el cliente quería, tenían diferentes opiniones. En vez de informarse sobre

lo que se quería, se comprometieron en una tercera interpretación. Cuando los ingenieros toman decisiones de diseño sin estar bien informados, a menudo, producen diseños poco prácticos y posiblemente no operativos.

Aunque todas estas causas son reales y contribuyen al número total de defectos de diseño, el mayor número de defectos proviene de simples descuidos, tonterías y malentendidos. La razón de que estas causas excedan en número a todas las demás, es porque, cuando los ingenieros software se enfrentan a un nuevo diseño del problema, normalmente tienen un cuidado especial. Saben que podrían fácilmente cometer un error y procuran comprobar su trabajo. Puesto que los ingenieros software cometen muchos errores simples y dichos errores pueden ser complicados de encontrar, estos simples defectos son una de las causas de muchos problemas. Dichos defectos logran pasar los procesos de diseño y pruebas y llegan a los usuarios del sistema. Estos defectos triviales causan muchos de los problemas que los usuarios tienen con los sistemas de software.

Resulta que muchos de estos defectos son evitables. El ingeniero sabía que esto era lo deseable. El diseño fue adecuadamente concebido, pero pobremente representado. Así, cuando llegó el momento de desarrollar el código, el programador no pudo ver lo que se pretendía. Antes que de tenerse y buscar al diseñador, muchos programadores completan el diseño precipitadamente sobre la marcha. Puesto que están trabajando en la implementación, a menudo no entienden todas las implicaciones del diseño. Así, es muy probable cometer un error. Aunque el diseñador sabía lo que quería, el programador no.

También puede haber un malentendido cuando estés implementando un diseño ideado por ti mismo. Cuando estás haciendo un diseño, a menudo buscarás un punto a partir del cual el resto del diseño parezca obvio. Si también estás planificando que harás el código, parece poco razonable documentar esta parte para el diseño. Desdichadamente, más tarde durante la implementación, puede que no recuerdes una cosa obvia del diseño y vuelvas a diseñar otra vez. Puesto que probablemente habrás olvidado el contexto del diseño, debes reconstruir todos los conceptos y condiciones relevantes del diseño. Puesto que este proceso de reconstrucción es propenso a errores, probablemente cometerás errores de diseño. La causa del error no fue una pobre implementación, sino que fue un diseño representado de manera muy pobre.

17.7

REPRESENTACIÓN DEL DISEÑO

Una clara y completa representación del diseño te ayudará a reducir el número de defectos introducidos. Esto significa que una vez que has realizado el diseño, necesitas representarlo para que esté claro para cualquier

persona que lo implemente. Esta es una cuestión crítica con programas de 10.000 LOC o más y puede ser un problema con módulos de programa de 50 a 100 LOC.

Representando completamente el diseño cuando lo crees, probablemente ahorrarás tiempo. Esto es debido a que puedes producir mucho más rápidamente a partir de un diseño completo y claro, que a partir de uno vago e incompleto. Puesto que un menor tiempo de codificación implica pocos defectos de codificación, con un buen diseño es más probable que hagas un programa de calidad superior.

Hay tres formas comunes de representar los diseños: gráficamente, con un seudocódigo o matemáticamente. En las siguientes secciones de este capítulo discutimos estos métodos de representación. Aunque esto es un tema muy extenso y se podría decir mucho sobre el mismo, este breve tratamiento te dará algunas ideas sobre el diseño de módulos de programas. También proporciona un contexto para pensar sobre el diseño, conforme utilizas el PSP en el futuro. El objetivo aquí, no es enseñar métodos de representación, sino convencerte de la importancia de hacer un diseño y de representarlo de forma clara. Así, cuando te enseñen varios métodos de representación en cursos o en el trabajo, podrás valorar por qué son importantes y tendrás en cuenta probarlos.

REPRESENTACIONESGRÁFICAS DEL DISEÑO

Las personas generalmente entienden las imágenes con más rapidez que las fórmulas o el texto. Cuando describes un diseño complejo, una imagen ilustrativa ayudará a las personas a entenderlo. Puesto que una gran parte del problema del diseño tiene que ver con la comprensión deberías, en general, utilizar representaciones gráficas para extender el diseño.

La forma más común de representación gráfica es el diagrama de flujo. Este es un diagrama con las funciones del programa representadas por cajas y la lógica del programa fluye representada por líneas que unen las cajas. Aunque hay muchas formas de dibujar dichos diagramas, describiremos los símbolos del diagrama de flujo tal y como se muestran en la Figura 17.1.

Un sencillo ejemplo de estos símbolos se muestra en la Figura 17.2. En la parte superior del diagrama, el símbolo conector muestra que la entrada x viene de $8Ax$ y en la parte inferior muestra que la salida y va a $3Ay$ y $5By$. Los símbolos en los conectores se refieren a otras páginas del diseño. Con esta notación, la variable x viene de la página 8 del diseño en el punto Ax . De forma similar, la variable y va a los puntos Ay de la página 3 y al By de la página 5.

La caja en la mitad de la Figura 17.2 muestra una función de decisión. Puedes representarla con una estructura `if -then -else` o con una

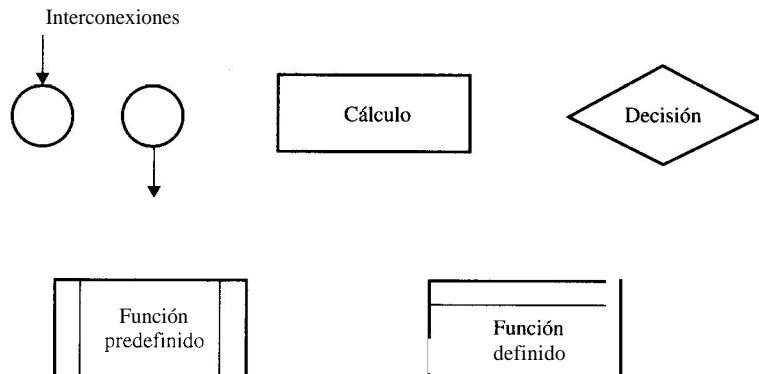


Figura 17.1 Símbolos de los diagramas.

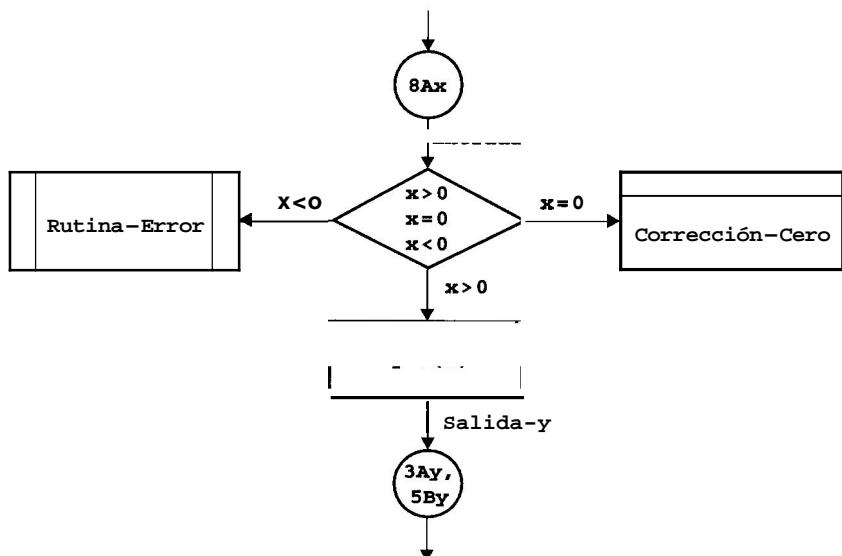


Figura 17.2 Ejemplo de diagrama lógico.

sentencia case. Esta elección se deja al programador. La caja que contiene, $y = f(x)$, describe un cálculo.

Las dos cajas a izquierda y derecha de la Figura 17.2 representan funciones definidas. A la izquierda tenemos la caja de la función, Rutina_Error, tiene dos barras verticales para representar una función predefinida. Esto se refiere a una función que ya ha sido implementada y es reutilizada en este programa. La caja de la derecha Corrección-Cero, tiene una línea horizontal para representar una función definida. Esta es una nueva abstracción funcional que estás definiendo y utilizarás en este programa cuando esté acabado.

Aunque las representaciones gráficas son fáciles de entender, a menudo, son imprecisas o voluminosas. Este no es un problema inherente al método, pero es otro ejemplo de una representación imprecisa e incompleta. Este problema de precisión puede ser tratado teniendo un diseño escrito completo y utilizando la representación gráfica para ayudar a explicar la lógica del programa. Recuerda, sin embargo, que cuanta más información pongas en el diagrama, mayor desorden habrá y más difícil será entenderlo. La mejor aproximación a este problema es utilizar varios niveles de diagrama de flujo, donde cada nivel incluye abstracciones que están definidas en diagramas de flujo de bajo nivel.

Debido a sus ventajas, deberías utilizar una representación gráfica para ilustrar el diseño de cada programa que hagas. A menudo, el proceso de dibujar el diagrama de flujo revelará relaciones que no habías considerado. A pesar de cómo lo hagas, dichos diagramas pueden ahorrar tiempo y reducir la confusión cuando posteriormente implementes, pruebas, corrillas o mejores el programa.

REPRESENTACIONES DEL DISEÑO CON SEUDOCÓDIGO

El seudocódigo proporciona otra forma de representar la lógica del programa. Aquí, la aproximación es escribir el programa en un lenguaje parecido al que utilizas para la implementación, pero con una escritura que sea fácilmente comprensible para las expresiones complejas. La idea es representar la lógica pero ignorar muchos de los requisitos sintácticos del lenguaje de programación. Un ejemplo de un seudocódigo para describir un diseño se muestra en la Tabla 17.3. Podrías extender esta descripción con sentencias adicionales para describir cómo se calcula la función $f(x)$. También podrías añadir definiciones de tipos, declaraciones u otros detalles si fuesen necesarios en el momento de la implementación.

Tabla 17.3 Ejemplo de representación en seudocódigo.

```
algoritmo (Función calculo f(x))
    if (x < 0)
        then (Rutina - Error)
    elseif (x = 0)
        then (Corrección - Nula)
    else
        Y = f (x)
    end.
```

Una descripción de diseño con seudocódigo es una mezcla de lenguaje humano y elementos de codificación. En vez de utilizar una lógica

completa para definir bien las funciones, se utilizan sentencias sencillas. De forma similar, el flujo del programa puede representarse con sentencias condicionales tradicionales, utilizando expresiones escritas que describen la lógica. Aunque no hay un estándar de notación de seudocódigo ampliamente aceptado, es una buena idea utilizar elementos del lenguaje que sean similares a aquellos que son utilizados en lenguajes de alto nivel. Cuando lo hagas, el seudocódigo parecerá familiar en el momento de la implementación y proporcionará un marco de trabajo para la misma.

Debido a que el seudocódigo no requiere los detalles del lenguaje de programación, es generalmente, más fácil y rápido para trabajar, que hacer un programa fuente completo. También, puedes documentar un diseño con muchos más detalles de los que necesites. Esto facilita la captura del diseño cuando desarrollas por primera vez. Esto es cuando entiendes mejor el diseño y puedes registrarla de una forma más eficiente.

La principal ventaja del seudocódigo es que es tan preciso como tú necesites y es fácil de entender. La principal desventaja es que, un diseño en seudocódigo puede ser tan detallado que sea difícil de entender. Por eso, es una buena idea proporcionar tanto el seudocódigo como un diagrama de flujo para cualquier diseño por muy sencillo que sea.

Un problema potencial con el seudocódigo es causado por el hecho de que omite mucha puntuación y estructura, utilizada para definir totalmente el significado de un programa fuente. Es fácil cometer errores con elementos lógicos comunes. Por ejemplo, es una buena idea utilizar la sangría o la puntuación para conectar cada sentencia `else` con el `if` apropiado, o mostrar la extensión de los bucles o condiciones `case`. El ámbito de las variables y los parámetros también puede causar confusión y debería ser cuidadosamente observado donde no es obvio.

Un problema común con el seudocódigo tiene que ver con el nivel de detalle. Muchos ingenieros escribirán una línea de seudocódigo por cada 3 o 5 líneas de programa fuente acabado. Sin embargo, he visto casos donde los ingenieros han hecho una línea de seudocódigo por cada línea de código fuente acabado. Aunque ellos pueden llamar a esto diseño, esto es realmente la implementación en una forma menos rigurosa. Puesto que el simple acto de copia introduce defectos, diseñando con tanto detalle se podría introducir más defectos que codificando desde el principio.

Debido a su flexibilidad, un diseño con seudocódigo puede estar en cualquier nivel de detalle. No hay forma de especificar con precisión qué debe contener un diseño con seudocódigo. Aunque las representaciones con seudocódigo pueden ser muy útiles, estamos aún con el problema que teníamos al principio: cómo definir con precisión lo que se entiende por diseño. En efecto, una gran ventaja del seudocódigo es que permite al ingeniero especificar el nivel de detalle del diseño necesario para cada situación.

Cuando estés utilizando el seudocódigo, los datos del PSP pueden ayudarte a decidir un nivel adecuado de detalle. Cuando se estén controlando datos de defectos, por ejemplo, si encuentras que estás introduciendo defectos del tipo diseño durante la fase de codificación, piensa la utilización de un diseño más preciso. A la inversa, si no has tenido problemas de representación del diseño, puedes intentar utilizar un seudocódigo menos detallado para ver si aumentas la velocidad en tu trabajo de diseño, sin causar errores de diseño en la implementación.

OTROS MÉTODOS DE REPRESENTACIÓN

Varios métodos matemáticos han sido ideados para definir con precisión sistemas software. Estos métodos tienen la ventaja de ser precisos y la desventaja de ser muy difíciles de aprender, al menos por ingenieros que no han tenido una formación adecuada en matemáticas. Los métodos formales, sin embargo, ofrecen una gran promesa de ayudar a los ingenieros a hacer programas con un mínimo número de defectos [Gries].

Puesto que la representación del diseño es un problema importante, indudablemente habrá muchas ideas nuevas y aproximaciones. Si consideras utilizar varias representaciones, ten en cuenta los siguientes puntos:

- Diseñar es un proceso mental.
- Una notación de diseño rica, puede ayudar a pensar con precisión y a representar un diseño complejo.
- Las notaciones ricas, sin embargo, son difíciles de aprender.
- Cuando estés utilizando una notación de diseño con la que no estés familiarizado, probablemente no pensarás en esa notación.
- Entonces debes pensar todo el diseño en una notación familiar y traducirla mentalmente a la notación no familiar.
- Este proceso de traducción inhibe la creatividad, retarda el trabajo de diseño y causa errores.

Cuando pruebas nuevos métodos de diseño y notaciones, intenta utilizarlos con fluidez antes de evaluarlos. También recuerda que la notación de diseño es un medio de comunicación. Si los implementadores no están a gusto con la notación, tendrán muchos problemas descritos anteriormente.

RESUMEN

La fuente de muchos defectos es la fase de codificación. Una forma de reducir los defectos de diseño es analizar los tipos de defectos. Defi-

nimos los defectos de diseño como los tipos de defectos del PSP del 50 al 100. Si introduces muchos de estos tipos en la fase de codificación, considera dedicar más tiempo al diseño. Es importante hacer el diseño lógico y funcional en la fase de diseño y no durante la codificación.

Las representaciones de diseño precisas pueden ahorrar tiempo de implementación y reducir los defectos de diseño. Una pobre representación puede causar defectos. Las representaciones gráficas son fáciles de entender pero pueden ser imprecisas. Las representaciones con seudocódigo pueden estar a varios niveles de detalle y pueden utilizar un lenguaje que es similar al lenguaje de programación utilizado. Al seleccionar una representación del diseño, familiarízate con la representación antes de hacer una evaluación. Utiliza tus datos del **PSP** para hacer la evaluación y tomar la decisión.

EJERCICIO 17

Como ejercicio, revisa los defectos que has encontrado en todos los programas de los que tengas datos de defectos. Utilizando los métodos mostrados en las Tablas 17.1 y 17.2, muestra cuál de estos defectos provienen de una representación imprecisa del diseño y sugiere los pasos para prevenirlos. Para el trabajo de casa, lista el número de defectos de diseño por tipos de problemas y la representación que podría haberlos previsto. También indica el tiempo total de corrección que esta mejora podría ahorrar.

Haz un seudocódigo y un diagrama de flujo para el próximo programa. Entrega los diseños con el programa, y escribe un breve comentario del impacto que estos métodos de diseño han tenido en el número de defectos de diseño introducidos. Muestra **los** datos que llevan a estas conclusiones. Obsérvese que tanto los análisis de defectos y los ejercicios de diseño son opcionales y no es necesario terminarlos a no ser que lo pida el profesor.

Entrega copias del Cuaderno de Registro de Tiempos y de las hojas del Resumen Semanal de Actividades que no hayas entregado previamente. También, entrega copias completas del Resumen del Plan del Proyecto, del Cuaderno de Registro de Defectos y de la Lista de Comprobación para la Revisión de Código para cada programa que desarrolles. Incluye los valores planificados, reales y Hasta la Fecha, de los tiempos de desarrollo, los defectos introducidos, los defectos eliminados y el rendimiento.

REFERENCIAS

[Curtis] Curtis, Bill, Herb Krasner, and Neil Iscoe. “A Field Study of the Software Design Process for Large Systems”, *Communications of the ACM*, November 1988, vol. 31, no. 11.

[Gries] Gries, David. *The Science of Programming*. New York: Springer-Verlag, 1981.

CAPÍTULO 18

Calidad del producto

Este capítulo discute cómo tu disciplina personal de trabajo afecta a la calidad de los productos que realizas. Ilustra la relación entre el número de defectos que encuentras durante la compilación y las pruebas, y el número de defectos que dejarás en tus productos acabados. Describe los pasos que puedes hacer para mejorar la calidad de tus programas. Como ejercicio, examinarás la relación, entre el número de defectos de compilación y el número de defectos de pruebas que tienes en los programas que has desarrollado en este curso.

18.1 LA CALIDAD VIENE PRIMERO

Como hemos visto, las pruebas son caras, aunque sea para pequeños programas. También hemos visto que puedes ahorrar tiempo encontrando defectos al principio. Revisando los programas antes de compilarlos y probarlos, reduce el número de defectos que encuentras en las pruebas y por lo tanto dedicarás menos tiempo a las mismas. También hemos visto que dedicando más tiempo a la revisión del código, tardarás menos en la compilación y en las pruebas. Además de estas ventajas, encontrando defectos al principio, haces productos de alta calidad. Este capítulo explica por qué esto es así.

Muchos ingenieros software piensan que ellos no deberían preocuparse por sus defectos hasta que comienzasen a compilar y probar sus pro-

gramas. Pasan deprisa por las etapas de diseño y codificación, y así pueden empezar las pruebas. Con el PSP, los ingenieros se centran en hacer programas limpios y libres de defectos desde el principio. La razón para esta estrategia, es que una vez que el ingeniero hace un producto poco sólido, no hay herramienta que pueda dejarlo libre de defectos. Una forma de demostrarlo, es mostrar los datos que relacionan los defectos encontrados en la compilación, con los encontrados en las pruebas y los defectos encontrados por los usuarios de los programas. Otra aproximación, sería que reunieses los datos de tu trabajo y vieses por ti mismo como esta estrategia trabaja para ti. Este capítulo se centra en ambas aproximaciones.

18.2 LASPRUEBAS

Cuento más complejo es el producto, las pruebas consumen más tiempo y son más caras. Las pruebas, también, son menos efectivas para encontrar los defectos. Es decir, con programas más grandes y complejos, será más costoso encontrar y corregir cada defecto, y los encontrarás en una tasa muy baja.

Hay varias razones para esto. Primero, los defectos enmascaran o agravan a otros. Es decir, cuando un programa tiene muchos defectos, sus interacciones complican el proceso de identificación y corrección del mismo. Esto hace que algunos de los defectos sean muy difíciles de encontrar y corregir. Un defecto también puede enmascarar los síntomas de otros. Puesto que estos defectos enmascarados son difíciles de encontrar, aumenta la posibilidad de que su detección se escape del todo.

Otra razón de la complejidad de las pruebas, es la dificultad de probar, aun en los programas pequeños, el número de caminos lógicos de un programa, ya que puede ser muy grande. No es práctico probar todas las combinaciones posibles ni tan siquiera en los programas pequeños. Conforme el programa es más grande y complejo, es más difícil hacer una prueba que sea moderadamente global. Considera el caso sencillo de un programa que analiza una cadena de caracteres. Si hay solamente 60 posibles caracteres, una cadena de 10 caracteres tendría 60^{10} posibles combinaciones. Es decir, un 6 seguido de 17 ceros. Si pudieras probar 1.000.000 de combinaciones en un segundo, necesitarías 20.000 años para probar todas las posibilidades. Así, vemos que no es posible probar todas las combinaciones imaginables.

En el anterior ejemplo, muchas de las combinaciones de caracteres serían lógicamente idénticas, pero los encargados de las pruebas del sistema no podrían saber cuáles ignorar, y cuáles probar. Los diseñadores, naturalmente, podrían decirles qué combinaciones deberían ignorar. El

problema de esta estrategia es que los diseñadores cometen errores. Si ellos no lo hicieran, naturalmente que no habría necesidad de probar sus programas. Las pruebas son un gran problema, pues los programas no se comportan de la forma que nosotros pensamos que lo harían. Así, con sistemas complejos, cuando únicamente probamos las condiciones que pensamos que son importantes, generalmente pasamos por alto muchos defectos.

Si sigues métodos disciplinados y revisiones cuidadosas, y pruebas tus pequeños módulos de programa, serás mucho más efectivo que cualquier grupo encargado de pruebas para encontrar defectos en tus programas. Esto se debe a que tú conoces la lógica del programa y qué harías. También comprendes las cosas inusuales y las condiciones singulares, y rápidamente puedes ver qué funcionará y qué no. Si haces todo lo que puedes para asegurar que los módulos de tu programa no tengan defectos, consigues eliminar en gran parte el coste y el tiempo requerido para anular los defectos posteriormente. Aunque no puedes evitar un error ocasional, puedes al menos asegurarte de que tus productos están esencialmente libres de errores. Y si tú no haces un programa libre de defectos, nadie *puede* hacerlo por ti.

18.3 EL FILTRO DE LAS PRUEBAS

Piensa en la eliminación de defectos como un filtro. Esto es, cada revisión, compilación y prueba elimina algún porcentaje de los defectos que están en el producto. Así, si pones más defectos en el filtro, probablemente encontrarás más defectos. Cuando, por ejemplo, pruebas un programa que contiene 25 defectos, tú esperas encontrar una parte de ellos, por ejemplo 12. De forma similar, si pruebas otro programa con 50 defectos, esperas encontrar una fracción parecida, unos **24**. Cuanto mayor es el número de defectos a encontrar, mayor es el número que probablemente encuentres.

Lo contrario de esto es, sin embargo, que cada proceso de eliminación de defectos también pasa por alto una fracción de los mismos. Así, a mayor número de defectos en el filtro, mayor es el número que probablemente serán pasados por alto. El número de defectos que se cogen por medio del filtro es por lo tanto, proporcional al número de defectos que entran al filtro. Aunque el grado que esta relación guarda, varía considerablemente con la calidad y tipo de proceso de eliminación de defectos, normalmente esta relación se mantiene. Así, a mayor número de defectos que entran en la fase de pruebas, compilación o revisión, mayor será el número con probabilidad de dejarlos en el producto a la salida de dichas fases.

La siguiente cuestión, naturalmente, tiene que ver con la calidad de los filtros. Si pudiésemos idear filtros con rendimientos de eliminación de defectos próximos al 100%, nuestro problema de eliminar defectos se resolvería. Desafortunadamente, los datos sobre los rendimientos de compilación y pruebas no son tranquilizadores. Como se muestra en la Tabla 18.1, las revisiones de código e inspecciones tienen mejores rendimientos, mientras la compilación, pruebas de unidad y otras formas de pruebas son menos efectivas [Humphrey 89]. Estas cifras están basadas en datos limitados y puede que no se apliquen a tu situación particular, pero estos son todos los datos que tenemos. La mejor respuesta para ti, naturalmente, sería reunir los datos de rendimiento de tus propios métodos de eliminación de defectos y sacar tus propias conclusiones. Es interesante observar que el método de mayor rendimiento de la Tabla 18.1 es para los ingenieros que hacen una revisión de código. El siguiente mayor rendimiento es para las inspecciones, donde varios ingenieros revisan entre sí el diseño y el código.

Tabla 18.1 Rendimientos de eliminación de defectos.

Método	Rendimiento aproximado (%)
Revisión de código	70-80
Inspección de código	50-70
Compilación	50
Prueba de unidad	40-50
Prueba de integración	45
Prueba de requisitos	45
Prueba del algoritmo	8

Los métodos que tienen los mayores rendimientos son manuales y no implican ninguna herramienta automatizada. La razón de que sean mejores que otros métodos es que la mente humana es el instrumento de detección de defectos más poderoso que cualquier herramienta software actual.

La conclusión lógica de estos datos es que, para hacer programas de alta calidad, debes tener el menor número de defectos posible al comenzar las pruebas. Comenzar las pruebas con el menor número de defectos, significa salir de la fase de compilación con el menor número de defectos. Finalmente, para salir de la fase de compilación con el menor número de defectos, debes eliminar los defectos antes de comenzar a compilar. Naturalmente, para hacer productos de máxima calidad, deberías medir, analizar y mejorar cada fase de eliminación de defectos.

18.4 LOS BENEFICIOS DEL TRABAJO CUIDADOSO

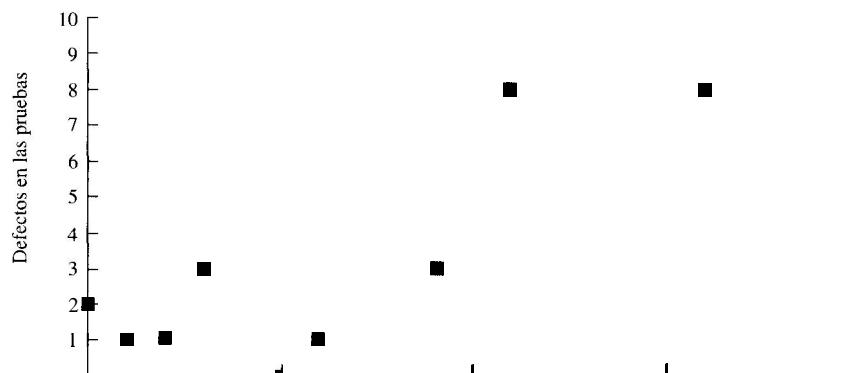
Otra forma para establecer el filtro de las pruebas es: “una chapuza, siempre es una chapuza”. En esencia, cuando los ingenieros hacen productos software poco sólidos, introducirán y dejarán más defectos en cada fase de compilación y pruebas. Cuando los ingenieros hacen el trabajo con esmero, los beneficios destacan en todo el proceso. Esto significa, por ejemplo, que cuando tienes muchos defectos en la compilación, probablemente tendrás muchos defectos en las pruebas. Y con muchos defectos en las pruebas, probablemente también tendrás muchos en el programa.

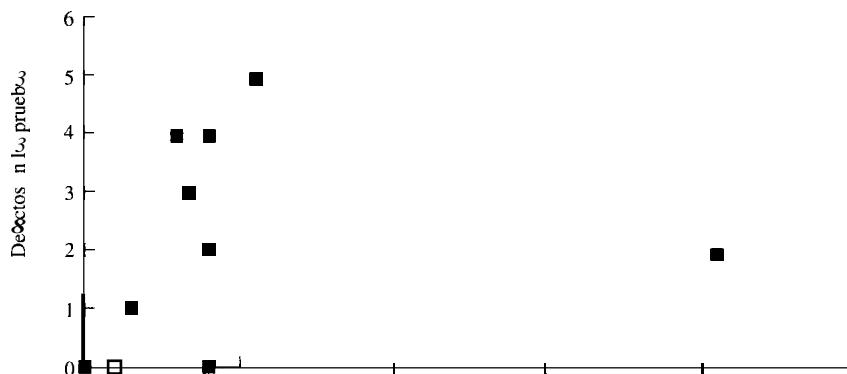
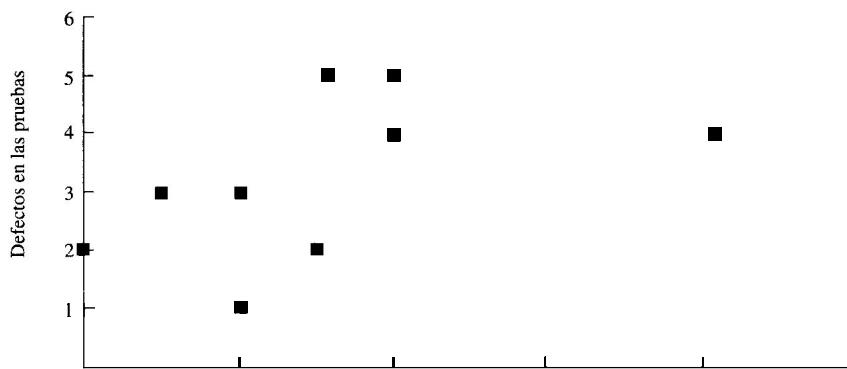
Como muestran los datos de las Figuras 18.1, 18.2 y 18.3, hay una relación general entre el número de defectos encontrados en la fase de compilación y los encontrados en la de pruebas. Aunque la relación no es fuerte en términos estadísticos, un gran número de defectos en la compilación indica posibles problemas en las pruebas.

18.5 EL CÁLCULO DE LOS VALORES DE RENDIMIENTO

El proceso de medida del rendimiento introducido en el Capítulo 16, tiene que ver con la tasa de eliminación de defectos antes de la primera compilación. La medida del rendimiento, sin embargo, puede ser aplicada a cualquier paso de la eliminación de defectos. Así, el rendimiento de cada fase puede calcularse de la siguiente forma:

Rendimiento de la Fase = $100 * (\text{número defectos eliminados durante la fase}) / (\text{número de defectos en el producto al inicio de la fase})$.





Un cálculo preciso del rendimiento requiere datos de defectos de todas las subsiguientes fases del proceso. Por ejemplo, si encuentras 5 defectos en una revisión de código, 3 en la compilación y 2 más en las pruebas, en la revisión de código habrás encontrado 5 defectos de un total de 10. Esto da un rendimiento de revisión del 50%. Aunque aún puede haber defectos en el programa, este es el rendimiento de la revisión en este punto.

Como se muestra en la Figura 18.4, cuando encuentres subsiguientes defectos, el rendimiento de todas las fases que pasan por alto esos defectos disminuye. A la salida de la revisión de código, habías encontrado 5 defectos. Puesto que estos son todos los defectos que has podido encontrar, estos son todos los que crees que hay. A la salida de la revisión, creías que el rendimiento de la misma era del 100%.

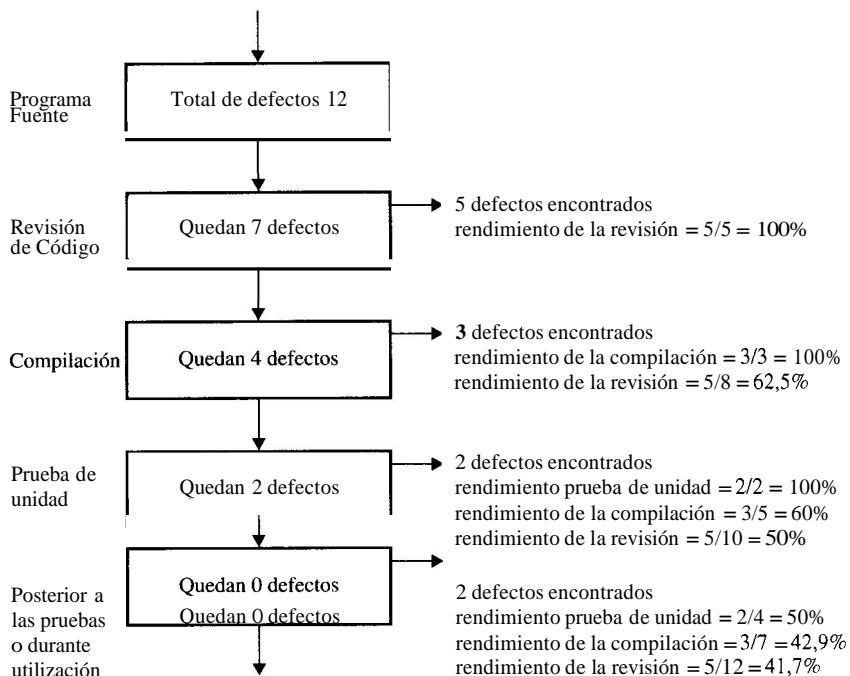


Figura 18.4 Valores de rendimiento.

Si en la fase de compilación encuentras 3 defectos más, sabemos ahora que en la fase de revisión se encontraron solo 5 defectos de un total de 8. Así, el rendimiento de la revisión fue realmente del 62,5% en vez del 100%. El rendimiento aparente del compilador es del 100%.

Si en la prueba de unidad encuentras 2 defectos más, en la revisión se habría encontrado solamente 5 de un total de 10 defectos. Así, el rendimiento de la revisión fue realmente del 50%. En este punto el compilador encontró 3 defectos pero pasó por alto 2, así el rendimiento en ese punto es del 60%. Puesto que en las pruebas de unidad se supuso que se habían encontrado todos los defectos restantes, el rendimiento aparente es del 100%.

Supóngase que en la subsiguiente prueba del programa, encuentras 2 defectos más. Entonces en la revisión se habría encontrado 5 de un total de 12 defectos que tiene el producto. Así, el rendimiento de la revisión en este punto sería del 41,7%. De forma similar, el compilador había encontrado 3 de 7 defectos, por lo que su rendimiento en ese punto sería del 42,9%. Incluso ahora no podrías tener la certeza de los valores de rendimiento final. Una vez entregado el producto a los usuarios finales, se podrían encontrar aún muchos defectos, lo que reduciría el rendimiento de todas las fases de eliminación de defectos.

La medida del rendimiento se puede aplicar a cualquier fase del proceso. Cuando se aplica a las pruebas de unidad, por ejemplo, se refiere a la tasa de defectos en el producto que son eliminados durante dichas pruebas. Como se describió en el Capítulo 16, el rendimiento del proceso se refiere a la tasa de defectos eliminados antes de la primera compilación. Así, el rendimiento del proceso se calcula de esta forma:

Rendimiento del Proceso = $100 * (\text{número de defectos eliminados antes de la compilación}) / (\text{número de defectos introducidos antes de la compilación})$.

18.6

LA ESTIMACIÓN DEL RENDIMIENTO DEFINITIVO

Aunque nunca puedes saber con certeza el rendimiento cuando acabas una fase, la medida de rendimiento ayudará a evaluar y mejorar el proceso. Por ejemplo, cuando encuentras 17 defectos en la revisión de código, 2 en la compilación y 1 en las pruebas, puedes asegurar de forma razonable que el rendimiento de la revisión estuvo próximo al 85%. Aunque posteriormente pueden encontrarse algunos defectos más, probablemente has encontrado ya la mayor parte de ellos. Si encuentras 17 defectos en la revisión, otros 15 en la compilación y 8 más en las pruebas, el rendimiento inicial de la revisión es del 42,5% y probablemente disminuirá en el futuro. Cuando encuentres 23 defectos en la compilación y pruebas, puedes asegurar de forma razonable que aún hay unos pocos defectos más en el producto.

La única forma de determinar cuántos defectos quedan, es controlar los que vas encontrando en el producto durante el resto de su vida útil.

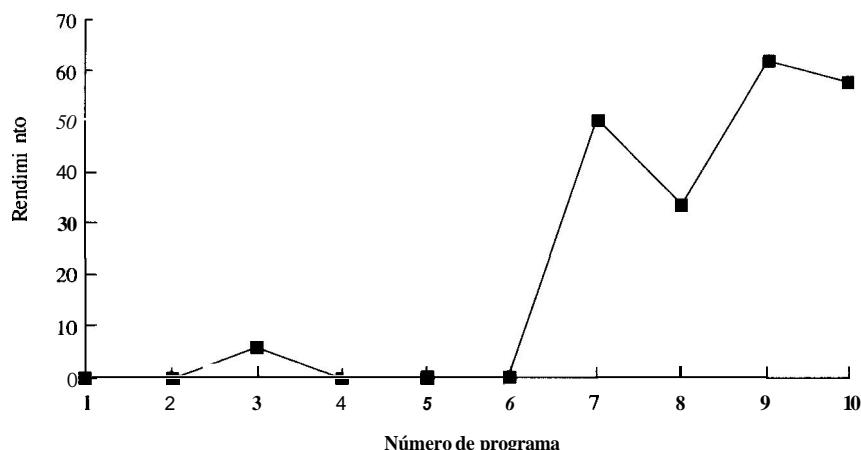


Figura 18.5 Mejora de rendimiento del Estudiante 1.

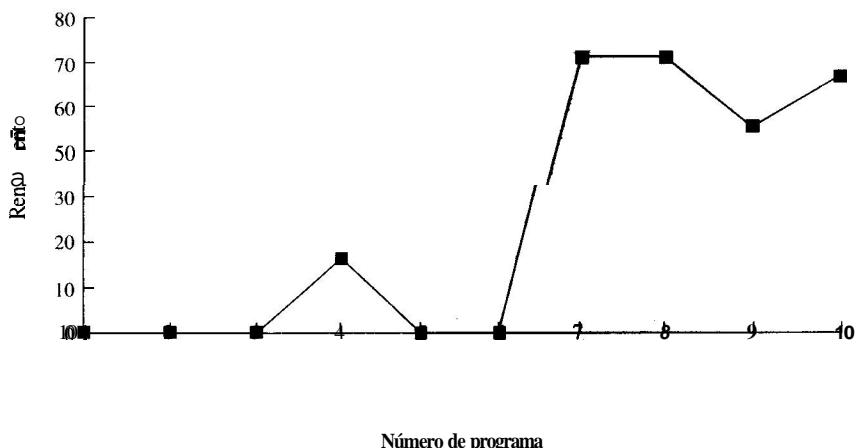


Figura 18.6 Mejora de rendimiento del Estudiante 14.

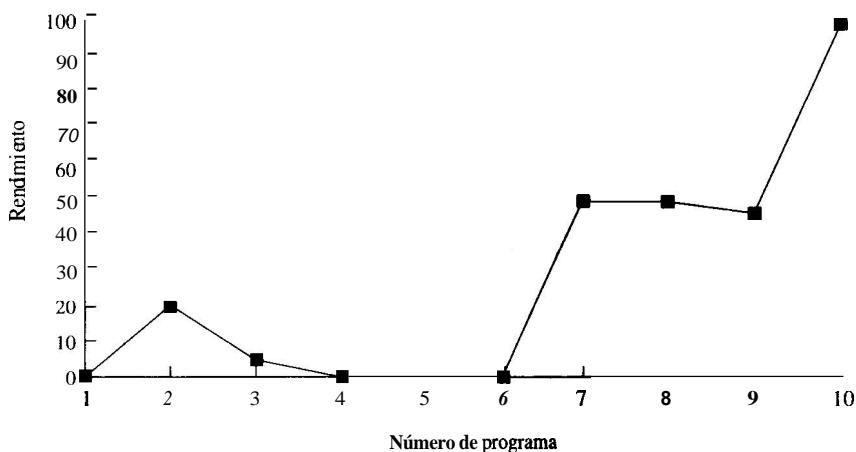


Figura 18.7 Mejora de rendimiento del Estudiante 20.

Debido a que durante los años de utilización es posible que no se encuentren todos los defectos, nunca tendrás la certeza. Si el número de defectos encontrados disminuye bruscamente con cada eliminación de defectos, los valores del rendimiento son probablemente bastante precisos. Después de que hayas reunido datos de defectos, puedes desarrollar medidas de rendimiento para cada fase. Entonces puedes calcular el número de defectos que quedan en cualquier producto que desarrolles.

Una regla de utilidad empírica es asumir que los defectos que quedan en un producto equivalen al número de los mismos encontrados en la última fase de eliminación. Esto equivale a asumir que el rendimiento de esta última fase fue del 50 %. Basándome en mis datos, este rendimiento es un poco bajo para las buenas inspecciones y revisiones de código, bas-

tante adecuado para la compilación y un poco alto para la mayor parte de las fases de pruebas.

Considera de nuevo el caso con los 17 defectos encontrados en la revisión de código, 2 en la compilación y 1 en las pruebas, la estimación actual del rendimiento de revisión es de $17/(17 + 2 + 1) = 85\%$. La regla empírica asumiría que se encontrará un defecto más. Esto daría un rendimiento final de $17/(17 + 2 + 1 + 1) = 80,95\%$. En el otro ejemplo, 17 defectos encontrados en la revisión de código, 15 en la compilación y 8 en las pruebas. Aquí, el valor del rendimiento actual es de $17/(17 + 15 + 8) = 42,5\%$. La regla de utilidad empírica sugiere que se encontrarán 8 defectos más, dando un rendimiento estimado definitivo de revisión de $17/(17 + 15 + 8 + 8) = 35,4\%$.

Con los datos históricos del rendimiento real para cada fase de eliminación de defectos, podrías hacer estimaciones definitivas de rendimiento más precisas. Harías esto utilizando el valor del rendimiento conocido para la fase final y calculando el número de defectos que probablemente se han pasado por alto. Con bastantes datos de rendimiento, podrías calcular rangos estadísticos para el número de defectos que quedan. Aunque nunca sabrás realmente los verdaderos valores del rendimiento, estos datos te ayudarán a hacer estimaciones un poco más aproximadas.

18.7

LOS BENEFICIOS DE UN RENDIMIENTO DE PROCESO DEL 100%

El objetivo de la revisión de código sería alcanzar de forma consistente un rendimiento de proceso del 100%. Si pudieras hacerlo, encontrarías todos los defectos y no dejarías ninguno para encontrarlo en la fase de compilación o pruebas. Esto no solamente ahorrará tiempo de compilación y pruebas, sino que el mayor beneficio estará en los costes y planificación del proyecto. Con módulos de programas libres de defectos, tu equipo de desarrollo no dedicaría tiempo a probar tus programas. Esto ahorraría los costes de las pruebas, reduciría el tiempo de las mismas y mejoraría las planificaciones del desarrollo. También produciría mejores productos.

Lograr un rendimiento de un 100% no es fácil. Requiere tiempo, práctica y un gran acopio de datos y análisis. El desarrollo de software de alto rendimiento, es una habilidad que puede aprenderse y mejorarse. Considera la analogía con tocar un instrumento musical. Cuando aprendes a tocarlo por primera vez, probablemente tocarás muchas notas erróneas. Con la práctica, irás gradualmente tocando mejor las notas y cometerás menos errores. Despues de mucha práctica, puedes aprender a

tocar sin cometer errores en las notas. Así, cuando se llega a esta situación, es cuando los músicos comienzan a ser realmente artistas.

El trabajo de realizar software de calidad es muy parecido a tocar un instrumento musical. Hasta que tú puedes hacer un código que tenga pocos o ningún defecto, no lograrás ser un ingeniero software. Mientras dediques la mayor parte de tu tiempo a encontrar y corregir defectos, no estarás actuando como un profesional. Con métodos de calidad disciplinados, sin embargo, harás pequeños programas libres de defectos de una forma coherente. Entonces, podrás centrarte en desafíos más importantes para producir grandes programas de alta calidad. Aunque dominar estas disciplinas requiere tiempo y un esfuerzo constante, verás que mejoras gradualmente. Tu objetivo personal, sin embargo, debe ser alcanzar un rendimiento del 100%. Recuerda que nunca harás grandes programas de calidad hasta que no hagas de forma rutinaria pequeños programas de calidad. Esto requiere disciplina personal y mucha práctica.

18.8

EXPERIENCIA DEL RENDIMIENTO

Tan difícil como alcanzar un alto rendimiento, es poder mantenerlo. Las Figuras 18.5 y 18.7 muestran las mejoras de rendimiento de tres ingenieros que lo hicieron. Estos ingenieros aprendieron el **PSP** en un curso donde escribieron un total de 10 pequeños programas. Comenzaron con el programa 1 y progresivamente añadieron medidas del **PSP** y métodos en los sucesivos programas. En el programa 7, se exigieron revisiones de código. Como se muestra en las figuras, todos los ingenieros habían mejorado claramente sus rendimientos.

Obsérvese, sin embargo, que no hay un patrón normalizado de mejora. Algunos ingenieros llevarán a cabo constantes mejoras, mientras que otros tendrán fluctuaciones considerables. Algunos estudiantes pueden alcanzar rendimientos del 80% o superiores, mientras que otros lucharán por conseguir el 50%. Durante mi desarrollo del **PSP**, aprendí a conseguir de forma consistente rendimientos del 80%. Esto, sin embargo, fue después de haber trabajado en la mejora del rendimiento durante tres años y haber escrito 62 programas con el **PSP**.

Las personas tienen diferentes habilidades. Algunas hacen de forma natural código de calidad mientras otras pueden visualizar diseños complejos. Algunas personas han nacido para hacer pruebas y otras son aptas para diseñar interfaces sencillas de usuario. Todos los ingenieros deberían ser capaces de alcanzar de forma consistente un rendimiento del 70%, pero algunos le llevará más tiempo que a otros.

18.9**PROTOTIPADO**

Una vez que has alcanzado de forma consistente un rendimiento del 70%, continuar mejorando será más difícil. Aunque ocasionalmente pases por alto uno o dos defectos sintácticos, la mayor parte de los defectos pasados por alto tendrán que ver con el diseño. Encontrarás nuevos sistemas o cuestiones de soporte o necesitarás utilizar funciones de programas desconocidas. Con grandes programas, las interfaces, el rendimiento, la gestión de memoria y muchos otros aspectos serán muy importantes.

Cada vez que hagas algo nuevo probablemente cometerás errores. Así, para eliminar todos los defectos, deberías intentar escribir prototipos de programas pequeños. Experimenta con cada función desconocida o procedimiento antes de utilizarlo en un programa. Prueba nuevas ideas y haz simples prototipos de cualquier estructura o elemento que no hayas utilizado previamente. Esto evitará muchos de los defectos que normalmente se escaparían a tus revisiones personales.

También encontrarás que ciertos tipos de defectos son más difíciles de localizar o prevenir que otros. Conforme ideas formas de encontrar los defectos con más frecuencia te equivocarás, muchos implicarán aplicaciones, soporte de sistemas y cuestiones del entorno de desarrollo. Cada fuente de confusión es una fuente de defectos. La clave está en reconocer la diferencia entre lo que realmente sabes de algo y lo que tú crees que sabes. Ocasionalmente cometerrás un error con algo que realmente conoces, pero muchas veces tus errores implicarán suposiciones que no son completamente correctas. Aprende a reconocer estas suposiciones. Entonces construye un prototipo para probarlas antes de incorporar las suposiciones en el producto.

RESUMEN

Conforme los programas son más grandes, es más costoso encontrar y corregir los defectos. El proceso de eliminación de defectos es también menos efectivo. La estrategia para producir grandes programas de gran calidad es, en primer lugar, eliminar todos los defectos de los módulos que forman estos grandes programas.

La eliminación de defectos es un proceso de filtrado: ve cada fase de eliminación de defectos como un filtro. Cuantos más defectos se pongan en el filtro más se encontrarán. También, cuanto más defectos se pongan en el filtro, más *se* pasarán por alto. El rendimiento de muchas fases de prueba es menor del 50%. Así, para obtener un producto de alta calidad al final de una prueba, debes poner un producto de alta calidad al comienzo de la prueba.

Un trabajo concienzudo en cada paso de **tu** proceso será rentable y ahorrará tiempo. Si haces un programa mal, se encontrarán muchos defectos en la compilación y en cada subsiguiente fase de pruebas.

El rendimiento mide la calidad de la eliminación de defectos. El rendimiento del proceso se refiere a la tasa de defectos en el producto que son eliminados antes de la primera compilación. El rendimiento puede medir también la tasa de defectos en un producto que son **eliminados** en la fase de eliminación de defectos. Tu objetivo sería lograr el **100%** de rendimiento del proceso. Recuerda: no serás capaz de hacer grandes programas de mucha calidad hasta que no puedas hacer de forma constante pequeños programas de gran calidad. Esto supone una dedicación constante a la calidad, disciplina personal y mucha práctica.

Para lograr la máxima calidad en un programa, **haz** pequeños prototipos para probar cada suposición antes de incorporarla al producto. Aprende a reconocer la diferencia entre lo que crees que sabes y lo que realmente sabes. Cada vez que hagas una **suposición**, es probable que introduzcas un defecto.

EJERCICIO 18

Para el ejercicio, muestra la relación entre los defectos de la fase de compilación y de pruebas para todos los programas que **has** escrito en este curso. Muéstralos en gráficos como **los** de las Figuras **18.1**, **18.2** y **18.3**. Calcula el rendimiento del proceso para cada programa, **y** estima la probabilidad del rendimiento final una **vez** que todos los defectos que queden en el programa se hayan encontrado. Observa que **este** análisis de defectos es un ejercicio opcional y no es necesario terminarlo a **no** ser que lo pida tu profesor.

Entrega copias del Cuaderno de Registro de Tiempos **y** del Resumen Semanal de Actividades que no hayas entregado previamente. También, entrega acabado el Resumen del Plan de Proyecto del PSP, el Cuaderno de Registro de Defectos y la Lista de Comprobación **para** la Revisión de Código para cada programa. Incluye **los** valores Planificados, Reales y Hasta la Fecha de **los** tiempos de desarrollo, de los defectos introducidos, de los defectos eliminados y del rendimiento.

REFERENCIA

[Humphrey 89] Humphrey, W.S. *Managing the Software Process*. Reading, MA: Addison-Wesley, 1989.

CAPITULO 19

La calidad del proceso

Este capítulo describe la calidad del proceso y algunas medidas que puedes utilizar para evaluar la calidad de tu trabajo de desarrollo de software. También muestra como calcular y controlar estas medidas. Como ejercicio, calcularás una de estas medidas para los programas que hayas desarrollado en este curso y dibujarás una gráfica que muestre cómo han cambiado estas medidas.

19.1 MEDIDAS DEL PROCESO

La calidad de tus programas está determinada por la calidad de tu proceso. La calidad de tu proceso, por otra parte, está determinada por tu forma de trabajar. Así, para desarrollar mejores programas, necesitas cambiarla. Para hacerlo más eficientemente, necesitas saber cómo es de bueno tu proceso. Para saber esto, necesitas medir la calidad de tu proceso.

La medida fundamental de un proceso tiene que ver con el volumen de productos realizados, su calidad, el tiempo y los recursos requeridos para hacer el trabajo. A partir de aquí, puedes determinar tu rendimiento real y cómo puedes cambiar para obtener mejores resultados en el futuro. En los Capítulos 3-11, introdujimos varias medidas de coste y tamaño. En el Capítulo 12, se comenzó a controlar defectos y se introdujeron medidas de eliminación de defectos y el número de defectos eliminados por hora. En los Capítulos 16-18 se introdujo el rendimiento y otras medidas de la

efectividad del proceso en la búsqueda y corrección de defectos. Ahora, nos centramos en las medidas de calidad del proceso.

Es importante recordar que en este libro nos centramos solamente en la calidad desde la perspectiva de la introducción y eliminación de defectos. Después de haber aprendido y practicado los métodos del PSP descritos en este libro, deberías considerar cómo mejorar el proceso que utilizas en otras áreas importantes. Por ejemplo, ¿realmente has entendido los requisitos? Algún paso del proceso que lleva a requisitos equivocados podría incluir desarrollar un primer prototipo, para revisar con tu profesor o cliente. Otro tópico importante tiene que ver con la utilidad o rendimiento de las pruebas, la compatibilidad de estándares, la revisión de interfaces, etc. Para cada una de estas cosas, puedes idear varios pasos para mejorar la calidad del producto. La calidad es un tema importante y hay, potencialmente, muchas acciones útiles del proceso que podrías tener en cuenta para ayudarte a asegurar que entregas productos de calidad a tus clientes.

19.2

LA PARADOJA DE LA ELIMINACIÓN DE DEFECTOS

Un fenómeno que puedes haber observado, es que las tasas de eliminación de defectos disminuyen conforme mejora la calidad del producto. Esto es porque, por ejemplo, la tasa de eliminación media para los defectos de las pruebas de la Figura 16.2 disminuyó de 4 a 2 por hora en un rango de 10 programas. En este mismo período, los defectos que estos estudiantes encontraron en las pruebas disminuyó en un factor de 10: de 40 a 4 defectos/KLOC.

Esta disminución parece natural cuando consideras casos extremos. Cuando hay muchos defectos que localizar, normalmente encontrarás muchos rápidamente. Aunque un defecto ocasional pueda llevar mucho tiempo, el tiempo medio es relativamente corto. En el otro extremo, con un solo defecto en un gran programa, todo el tiempo dedicado a revisar y probar será asignado a este único defecto. Dependiendo de la naturaleza del defecto y del orden de las pruebas y revisiones, esto podría llevar mucho tiempo.

Así, con pocos defectos, llevará mucho tiempo encontrarlos, independientemente de los métodos que utilices. Esto es cierto para la revisión y las pruebas, pero lo es menos para la compilación. Aunque hay una leve tendencia descendente para la tasa de eliminación de defectos en la compilación, la naturaleza automática de los compiladores los hace menos sensibles a la densidad de defectos. La tendencia descendente para las revisiones y pruebas puede no ser evidente para secuencias cortas de da-

tos, pero la tasa para cualquier método dado, generalmente, disminuirá conforme disminuya el número de defectos.

Así, conforme pongas progresivamente mayor calidad en los programas por medio de sucesivas fases de integración y pruebas del sistema, sus defectos residuales serán progresivamente más difíciles de encontrar y corregir. Esto sugiere que cuanto más defectos encuentres, más importante es encontrarlos y corregirlos.

19.3

UNA ESTRATEGIA PARA LA ELIMINACIÓN DE DEFECTOS

Se podría argumentar que muchos de los defectos en los programas grandes están en el sistema y no en los módulos. Esto, sin embargo, no es cierto. Con pocas excepciones, los defectos en los grandes sistemas están en los módulos que lo constituyen. Estos defectos se pueden dividir en dos clases: aquellos que implican solamente un módulo y los que implican interacciones entre módulos.

Cuando tengas un alto rendimiento con el PSP, eliminarás casi toda la primera clase de defectos. La segunda clase, sin embargo, es mucho más difícil. La razón es que, los sistemas grandes y complejos, implican muchas interacciones que son difíciles de visualizar para los diseñadores. Una buena forma de abordar este problema es seguir una estrategia como la siguiente:

- Esfuérzate en desarrollar módulos con la máxima calidad posible.
- Haz inspecciones de todas las interfaces de módulos y sus interacciones. (Observa que una inspección es cuando un equipo de ingenieros revisa un producto. Las inspecciones se tratan brevemente en la Sección 13.11.)
- Inspecciona los requisitos para asegurarte que todas las funciones importantes son adecuadamente entendidas, diseñadas e implementadas.
- Inspecciona el sistema y el diseño del programa frente a **los** requisitos, para asegurarte que son tratados adecuadamente todos los requisitos clave.
- Haz unas pruebas de unidad exhaustivas después de que se haya inspeccionado el código.
- Haz una prueba de integración global.
- Haz pruebas a todo el sistema.

Excepto el primer paso, el resto están más allá del ámbito de tu proceso personal. Sin embargo, el primer paso depende de ti. Si tus módulos no

son de máxima calidad, el resto de pasos serán mucho menos efectivos. Entonces el proceso de desarrollo debe concentrarse en encontrar y corregir los defectos que tú y tus compañeros dejan en tus módulos. Esto también significa que el trabajo que los restantes pasos se supone que acabaron, realmente no fue hecho. En vez de asegurar que el sistema realiza adecuadamente las funciones que el usuario pretende, las pruebas del sistema y de integración se dedicarán principalmente a encontrar y corregir los defectos que quedan en los módulos.

Con módulos que inicialmente tienen una alta calidad, sin embargo, puedes seguir la estrategia anteriormente mencionada. Entonces tendrás una oportunidad razonable de hacer sistemas de alta calidad. Mejor que derrochar el tiempo intentando encontrar y arreglar los defectos de los módulos, la prueba de integración se puede enfocar en probar las interfaces entre los módulos. La prueba del sistema ahora podría tratar cuestiones críticas del sistema como el rendimiento, la funcionalidad, recuperación y seguridad. Hasta que los ingenieros no logren de una forma consistente rendimientos personales altos, dichas pruebas no se harán.

19.4 EL COSTE DE LA CALIDAD

Aunque pudieras dedicar cualquier cantidad de tiempo a buscar defectos, nunca estarías seguro de haberlos encontrado todos. Así, si tu único objetivo era hacer programas libres de defectos, podrías continuar haciendo revisiones y pruebas indefinidamente. Aunque dejases de hacerlas después de no haber encontrado defectos por un tiempo, podrías aun, no haberlos encontrado todos. Como ingeniero del software, sin embargo, necesitarás un equilibrio entre el tiempo dedicado y la calidad de los productos hechos. El Coste de la Calidad (CDC) proporciona una forma de tratar estas cuestiones. El CDC tiene tres elementos principales: costes de los fallos, costes de valoración y costes de prevención.

Los costes de los fallos incluyen todos los costes de corregir los defectos del producto. Mientras estás corrigiendo un defecto, estás incurriendo en unos costes de los fallos. De forma similar, mientras estás ejecutando el depurador para detectar la sentencia defectuosa, estás incurriendo en costes de fallos. Cualquier cosa hecha como una parte normal de la reparación de un defecto se contabiliza como coste de fallos. Esto incluye el volver a diseñar, a compilar y a probar.

Los costes de valoración incluyen todo el trabajo de valoración del producto para ver si tiene defectos, excluyendo el tiempo dedicado a la corrección de defectos. Incluye la revisión de código, el tiempo de compilación y las pruebas para un programa libre de errores. Así, los costes de

valoración no incluyen los costes de reparación. El coste de valoración es la tasa pagada para asegurar que el programa está libre de defectos.

Los costes de prevención son los costes incurridos cuando modificas el proceso para evitar introducir defectos. Incluye, por ejemplo, los análisis hechos para comprender los defectos. También incluye el trabajo realizado para mejorar los procesos de especificación de requisitos, diseño e implementación. El tiempo gastado en el rediseño y pruebas de un nuevo proceso, es también un coste de prevención.

Otra actividad importante para la prevención de defectos es la construcción de prototipos para probar el diseño o las ideas de implementación. Cuando estás utilizando una función de librería desconocida puedes estar tentado a tan solo usarla. Sabes, sin embargo, que esto conduce a errores de diseño y defectos en el programa. El coste de escribir pequeños prototipos del programa para comprobar cómo trabaja dicha función es un coste de prevención.

19.5

CÁLCULO DEL COSTE DE LA CALIDAD

El PSP mide el CDC de una forma muy sencilla. Aunque el tiempo dedicado a la compilación incluye algún tiempo de compilación libre de defectos, el PSP contabiliza todo el tiempo de compilación como costes de fallos. De forma similar, las pruebas incluyen algún tiempo que debería dedicarse a probar un programa libre de defectos. De nuevo, el PSP contabiliza todo el tiempo de pruebas como costes de fallos. Finalmente, todo el tiempo de revisión es contabilizado como coste de valoración. Este tiempo incluye algún coste de reparación, pero el PSP contabiliza todo el tiempo de revisión como coste de valoración. Podrías calcular los valores exactos del CDC con los métodos descritos en la Sección 19.8, pero esto implica mucho trabajo y no cambia considerablemente la efectividad de las medidas. Por eso, recomiendo que utilices las definiciones simplificadas del PSP.

El coste de la calidad se calcula como un porcentaje del tiempo de desarrollo total. Para el PSP, los costes de valoración y fallos se calculan de la siguiente forma:

- La valoración CDC es la suma de todo el tiempo de revisión como un porcentaje del tiempo total de desarrollo.
- Los fallos CDC es la suma de todo el tiempo de compilación y pruebas como un porcentaje del tiempo total de desarrollo.

Por ejemplo, supongamos que tienes los datos del proceso que se muestran en la Tabla 19.1 Resumen del Plan del Proyecto. Podrías calcular la valoración CDC de la siguiente forma:

Tabla 19.1 Resumen del plan del proyecto del PSP.

Estudiante	Estudiante X		Fecha	9/12/96
Programa			Programa#	15
Profesor	Sr. Z		Lenguaje	Ada
Resumen	Plan	Real	Hasta la Fecha	
Minutos/LOC	5,48	4,60	5,35	
LOC/Hora	10,95	13,04	11,21	
Defectos/KLOC	92,53	52,6	86,7	
Rendimiento	40,0	100,0%	45,5	
V/F	0,375	1,93	0,44	
Tamaño Programa (LOC):				
Total Nuevo & Cambiado	49	57	392	
Tamaño Máximo	62			
Tamaño Mínimo	36			
Tiempo por Fase (min.)	Plan	Real	Hasta la Fecha	% Hasta la Fecha
Planificación	17	20	140	6,7
Diseño	29	30	233	11,1
Codificación	116	119	911	43,4
Revisión del código	21	29	174	8,3
Compilación	15	5	105	5,0
Pruebas	41	10	209	13,8
Postmotten	30	41	247	11,7
Total	269	262	2099	100,0
Tiempo Máximo	340			
Tiempo Mínimo	197			
Defectos introducidos	Plan	Actual	Hasta la Fecha	% Hasta la Fecha
Planificación				
Diseño	1	5	14,7	1,29
Codificación	4	3	82,4	1,84
Revisión del código				
Compilación			1	2,9
Pruebas				
Total	5	3	34	100,0
Defectos eliminados	Plan	Actual	Hasta la Fecha	% Hasta la Fecha
Planificación				
Diseño				
Codificación				
Revisión del código	2	15	44,1	5,17
Compilación	2	13	35,2	7,43
Pruebas	1	6	17,1	1,25
Total	5	34	100,0	

- Tiempo total real de desarrollo = 262 minutos.
- Tiempo real de revisión del código = 29 minutos.
- Valoración CDC = $100*29/262 = 11.07\%$.

Los costes de los fallos se obtendrían así:

- Tiempo real de compilación = 5 minutos.
- Tiempo real de pruebas = 10 minutos.
- Fallos CDC = $100*(5+10)/262 = 100*15/262 = 5,73\%$.

19.6**LA RELACIÓN VALORACIÓN /FALLOS**

Una medida útil del PSP es la relación Valoración/Fallos, o V/F. Es calculada dividiendo la valoración CDC por los fallos CDC. En el ejemplo de la Sección 19.5, con una valoración CDC de 11,07% y con un valor de fallos CDC de 5,73%, $V/F = 11,07/5,73 = 1,93$.

Es sencillo calcular el valor V/F como el tiempo de revisión de código dividido por el total del tiempo de compilación y pruebas. Utilizando esta aproximación, $V/F = 29/(5+10) = 1,93$, que es el mismo valor que hemos obtenido en el párrafo anterior. Estos cálculos están descritos en las Instrucciones del Resumen del Plan del Proyecto en la Tabla 19.2.

El valor de V/F mide la cantidad relativa de tiempo dedicada a encontrar defectos antes de la primera compilación. Los datos del estudiante X muestran que el valor V/F es un buen indicador de la probabilidad de encontrar defectos en las pruebas. En la Figura 19.1, los valores de V/F para 14 estudiantes del PSP se muestran frente al número de defectos/KLOC que encontraron en las pruebas. Estos datos se refieren a un total de 140 programas.

Cuando el valor de V/F es menor que 1, las pruebas del programa generalmente encuentran muchos defectos. Aunque esto no está garantizado, muchos programas en este rango tendrán un valor bastante alto de defectos/KLOC en las pruebas. También, de la Figura 19.1, se deduce claramente que procesos con V/F por encima de 2 tienen pocos defectos/KLOC en las pruebas. Esto sugiere que procesos con un valor de V/F por encima de 2 es más probable que produzcan productos libres de defectos que los procesos que tienen un valor de V/F por debajo de 1. Deberías intentar lograr unos valores de V/F de 2 o superiores.

Los datos de una empresa muestran cómo puede ser de efectivo el valor de V/F para juzgar la calidad del producto. La Figura 19.2 muestra un gráfico del número de defectos encontrados en las pruebas de unidad de seis componentes de un producto frente al valor de V/F. La Figura 19.3 muestra el número de defectos encontrados durante la integración, las pruebas del sistema y la utilización por el cliente de estos mismos componentes. Obsérvese que uno de los componentes con un valor bajo de V/F tenía relativamente pocos defectos en las pruebas de unidad pero se encontraron muchos más tarde. Obviamente, las pruebas de unidad no habían sido muy minuciosas. La Figura 19.4 muestra los defectos encontrados en la compilación frente al número de defectos encontrados después de la compilación. Evidentemente, cuantos más defectos se encontraron en la compilación, más defectos se encontraron en las subsiguientes fases de pruebas y utilización. El mensaje es claro: si quieres reducir los defectos de las pruebas, debes limpiar tus programas antes de comenzar la compilación. El componente con el mayor número de defectos de compilación

Tabla 19.2 Instrucciones del resumen del plan del proyecto del PSP.

Propósito	Esta tabla trata los datos estimados y reales de los proyectos, de una forma cómoda y fácilmente recuperable.
Cabecera	Introduce los siguientes datos: <ul style="list-style-type: none"> • Tu nombre y fecha de hoy. • Nombre y número de programa. • Nombre del profesor. • El lenguaje que utilizarás para escribir el programa.
Minutos/LOC	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> • Escribe los Minutos/LOC planificados para este proyecto. Utiliza la velocidad Hasta la Fecha de un programa reciente del Cuaderno de Registro de Trabajos o del Resumen del Plan del Proyecto. Después del desarrollo: <ul style="list-style-type: none"> • Divide el tiempo total de desarrollo por el tamaño real del programa para obtener los Minutos/LOC reales y los Minutos/LOC Hasta la Fecha. • Por ejemplo, si el proyecto se hizo en 196 minutos e hiciste 29 LOC, los Minutos/LOC serían $196/29 = 6,76$.
LOC/Hora	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> • Calcula las LOC por hora planificadas para este programa dividiendo 60 por los Minutos/LOC de la casilla de Plan. Después del desarrollo: <ul style="list-style-type: none"> • Para LOC/Hora Real y Hasta la Fecha divide 60 por los Minutos/LOC Reales y Hasta la Fecha. • Para los 6,76 Minutos/LOC Reales, tenemos $60/6,76 = 8,88$ LOC/Hora Reales.
Defectos/KLOC	Antes de desarrollar: <ul style="list-style-type: none"> • Encuentralos defectos/KLOC Hasta la Fecha del programa más reciente. • Utiliza dicho valor como los Defectos/KLOC planificados. Después del desarrollo: <ul style="list-style-type: none"> • Calcula los valores de defectos/KLOC reales y Hasta la Fecha para este programa. • Para el valor real, multiplica el total de defectos reales por 1000 y divídelo por las LOC Total Nuevas & Cambiadas Reales. • Para el valor Hasta la Fecha haz unos cálculos similares. • Con 17 defectos Hasta la fecha y 153 LOC Total Nuevas & Cambiadas, los Defectos/KLOC Hasta la Fecha = $1000*17/153 = 111,11$.
Rendimiento	Calcula el rendimiento planificado, real y Hasta la Fecha. Rendimiento = $100 * (\text{número de defectos eliminados antes de la compilación}) / (\text{número de defectos introducidos antes de la compilación})$, así, con 5 defectos introducidos y 4 localizados, el rendimiento = $100 * 4/5 = 80,0\%$.
V/F	Calcula el valor de V/F planificado, real y Hasta la Fecha. Para el valor real, por ejemplo, toma la relación del tiempo de revisión de código real y divídelo por la suma de los tiempos reales de compilación y pruebas. Para un tiempo de revisión de 29 minutos, uno de compilación de 5 minutos y uno de pruebas de 10 minutos, $V/F = 29/(5 + 10) = 1,93$.

(Continúa)

Tabla 19.2 Instrucciones del resumen del plan del proyecto del PSP.
(Continuación)

Tamaño Programa (LOC)	Antes de iniciar el desarrollo: <ul style="list-style-type: none"> Escribe bajo la columna plan, el valor estimado de LOC Total Nuevas & Cambiadas, Máximo y Mínimo. Después del desarrollo: <ul style="list-style-type: none"> Cuenta y anota las LOC Nuevas & Cambiadas Reales. Para la columna Hasta la Fecha, añade LOC Reales Nuevas & Cambiadas a las LOC Hasta la Fecha Nuevas & Cambiadas de programas anteriores.
Tiempo por Fase Plan	Para el tiempo total de desarrollo, multiplica el valor de las LOC Total Nuevas & Cambiadas por Minutos/LOC. Para el tiempo Máximo, multiplica el tamaño Máximo por los Minutos/LOC. Para el tiempo Mínimo, multiplica el tamaño Mínimo por los Minutos/LOC. Del Resumen del Plan del Proyecto de un programa reciente, busca los valores de % Hasta la Fecha para cada fase. Utilizando el % Hasta la Fecha de programas anteriores calcula el tiempo planificado para cada fase.
Actual	Una vez acabado el trabajo, anota el tiempo real en minutos que has gastado en cada fase del desarrollo. Obtén estos datos del Cuaderno de Registro de Tiempos.
Hasta la Fecha	Para cada fase, anota la suma del tiempo real y el tiempo Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, anota 100 multiplicado por el tiempo Hasta la Fecha y lo divides por el total del tiempo Hasta la Fecha.
Defectos introducidos Plan	Antes del desarrollo, estima el número total de defectos a introducir en el programa. El valor es: defectos/KLOC planificados multiplicado por las LOC Total Nuevas & Cambiadas planificadas para este programa y dividido por 1000. Por ejemplo, con un valor de defectos/KLOC planificado de 75,9 y un valor de LOC Nuevas & Cambiadas planificadas de 75, el total de Defectos Planificados = $75,9 * 75 / 1000 = 5,69$, redondeo a 6. Antes de desarrollar, estima los defectos introducidos por fase utilizando el total de defectos estimados y el % Hasta la Fecha de defectos introducidos de programas anteriores.
Real	Después del desarrollo, localiza y anota el número real de defectos introducidos en cada fase.
Hasta la Fecha	Para cada fase, anota la suma de los defectos reales y los defectos Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, multiplica por 100 los defectos Hasta la Fecha para esa fase y divídelos por el total de defectos Hasta la Fecha.
Defectos/hora	Calcula los defectos introducidos por hora para la fase de diseño y codificación. Para la fase de diseño, por ejemplo, se multiplica 60 por los defectos de diseño Hasta la Fecha y se divide por el tiempo de diseño Hasta la Fecha = $60 * 5 / 195 = 1,54$ defectos/hora.

Tabla 19.2 Instrucciones del resumen del plan del proyecto del PSP.
(Continuación)

Defectos eliminados	En la fila de total, anota el total de defectos estimados.
Plan	Utiliza los valores de % Hasta la Fecha de los programas más recientes, calcula los defectos eliminados planificados para cada fase.
Real	Después del desarrollo, localiza y anota el número real de defectos eliminados en cada fase.
Hasta la Fecha	Para cada fase, anota la suma de los defectos reales y los defectos Hasta la Fecha de los programas más recientes.
% Hasta la Fecha	Para cada fase, multiplica por 100 los defectos Hasta la Fecha para esa fase y divídelos por el total de defectos Hasta la Fecha.
Defectos/hora	Calcula los defectos eliminados por hora para la revisión de código, compilación y pruebas. Para las pruebas, por ejemplo, multiplica 60 por los defectos de las pruebas Hasta la Fecha y divídalo por el tiempo de pruebas Hasta la Fecha = $60*6/279 = 1,29$ defectos/hora.

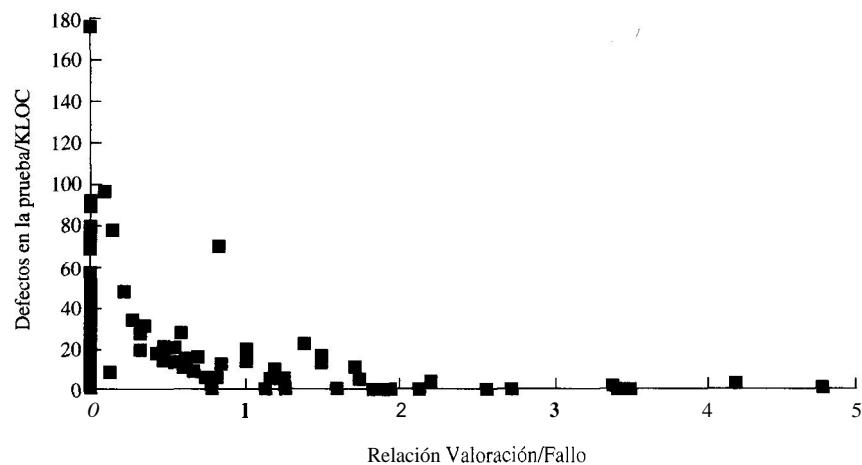


Figura 19.1 Defectos en la prueba vs. V/F.

tenía más defectos en cada subsiguiente fase del proceso. También tenía más defectos en el producto entregado al cliente.

Para mostrar el impacto de una pronta eliminación de defectos en el tiempo de desarrollo, la Figura 19.5 muestra las horas de pruebas frente a los valores de V/F. Aquí, los dos componentes con valores bajos de V/F tuvieron 140 y 245 horas de pruebas, mientras que los otros tuvieron de 10 a 40 horas de pruebas. La Figura 19.6 muestra las horas de pruebas por KLOC. En un proceso de baja calidad, las pruebas consumen de 80 a 100 horas por KLOC; con un proceso de alta calidad, las pruebas suponen sobre unas 20 horas. Esta diferencia de 60 a 80 horas de pruebas por KLOC

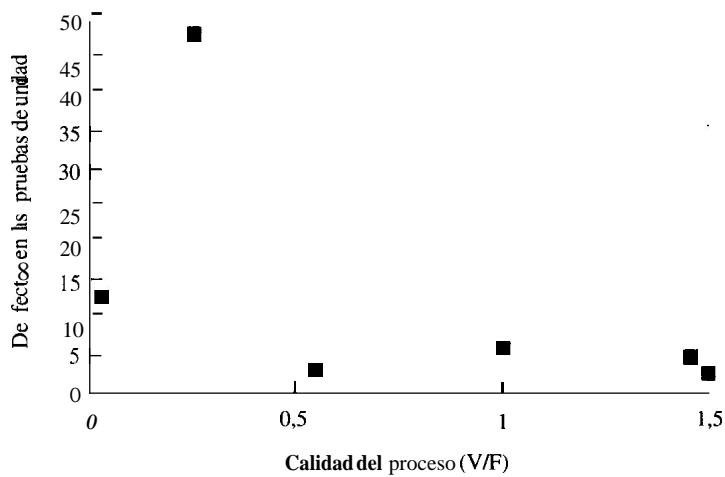


Figura 19.2 Defectos en la prueba de unidad vs. calidad del proceso.

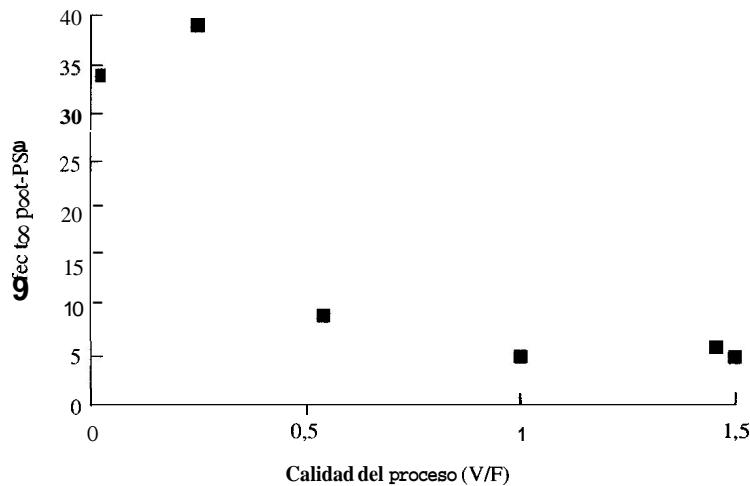


Figura 19.3 Defectos post-PSP vs. calidad del proceso.

puede ser muy significativa cuando trabajes con productos de 100.000 LOC o más. El coste añadido de un PSP de poca calidad estaría entre 6.000 a 8.000 horas adicionales de pruebas, o de 3 a 4 años de tiempo de programador derrochados. En productos de varios millones de LOC, que pronto serán comunes, esto podría ser la diferencia entre un producto entregado a tiempo y el fracaso total.

UTILIZANDO LA RELACIÓN VALORACIÓN / FALLOS

Para lograr valores de V/F por encima de 2,0 revisa el histórico de compilaciones, el tiempo de pruebas y planifica dedicar al menos el doble de

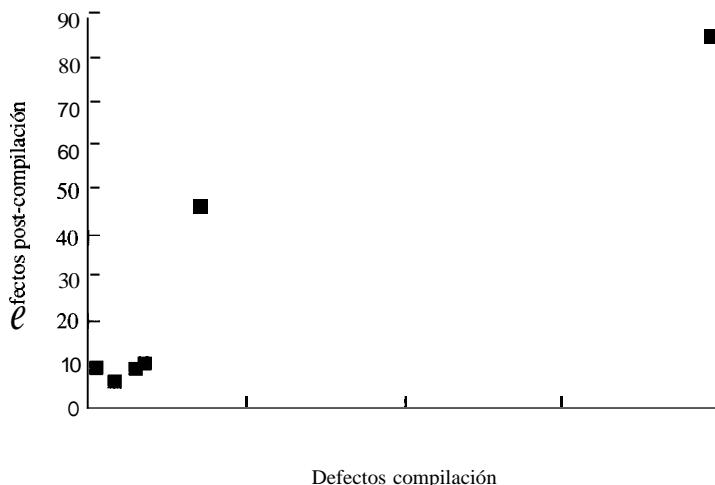
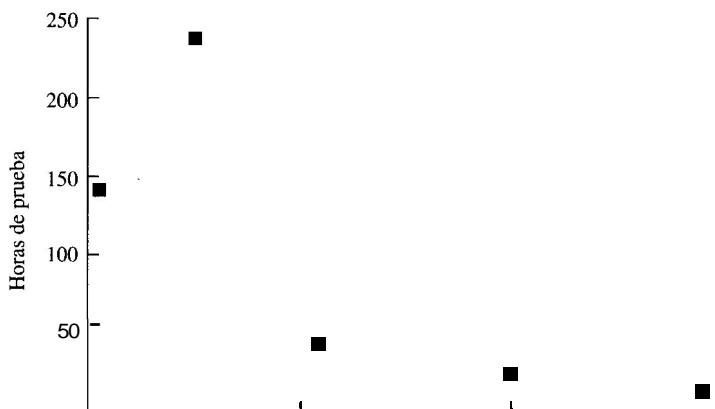


Figura 19.4 Defectos compilación vs. defectos post-compilación.



tiempo a la siguiente revisión de código. Puedes aumentar el valor de V/F simplemente dedicando más tiempo a la revisión de código. Si no encuentras defectos, no mejorará la calidad del programa. Es importante para la productividad utilizar el tiempo de revisión para encontrar defectos.

Mientras que tu rendimiento no esté en un rango del 80 al 100%, continúa aumentando el valor de V/F. No debes hacer esto, solamente dedicando más tiempo. Revisa **los** datos de los defectos de tus programas más recientes e idea formas de encontrar todos los defectos que con más fre-

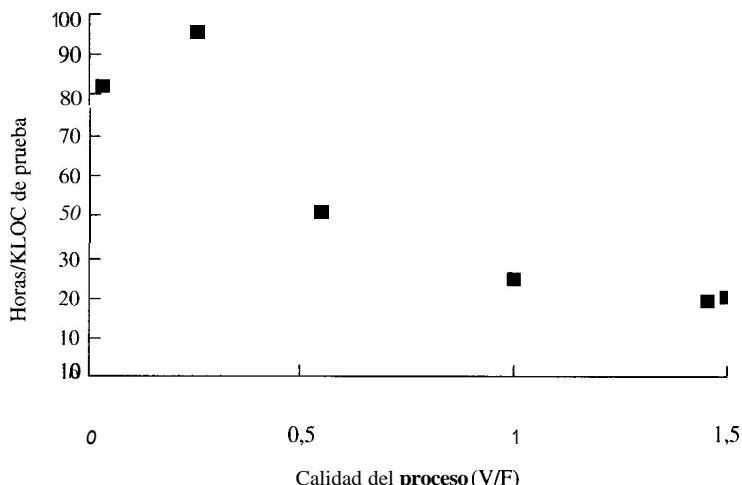


Figura 19.6 Horas/KLOC de prueba vs. calidad del proceso.

cuencia pasas por alto. Si no estás seguro de cómo hacerlo, relee el Capítulo 14.

Después de determinar cómo encontrar los defectos que pasas por alto con más frecuencia, introduce los pasos apropiados en la lista de comprobación para la revisión del código. Por último, sigue estos pasos cuando hagas la revisión de código. Si haces todo esto, dedicarás más tiempo a la revisión, encontrarás más defectos y aumentará el valor de V/F. Posiblemente reduzcas el número de defectos que encuentres en la compilación y en las pruebas. Esto ahorrará mucho tiempo de pruebas, reducirá costes de fallos y realizará productos de alta calidad.

19.7

CÓMO MEJORAR LAS TASAS DE REVISIÓN

No te preocupes de las tasas de revisión de código defecto/hora hasta que de forma regular encuentres casi todos los defectos antes de la compilación y las pruebas. Continúa comprobando los datos de defectos para ver qué pasos de la revisión deberían haber encontrado los defectos que se encontraron en la fase de compilación y en la de pruebas. También, continúa actualizando tu lista personal de comprobación para la revisión de código para encontrar estos defectos.

Una vez que encuentres la mayor parte de los defectos en la revisión de código, piensa cómo mejorar las tasas de revisión de código. Para hacer esto, identifica cualquier paso de la revisión que ni encuentre ni pase por alto defectos. A continuación, reconsidera tus razones para incluir estos pasos en primer lugar. Si estas cuestiones ya no son un problema, sál-

Tipos de defectos			
10 Documentación	50 Interfaz	90 Sistema	
20 Sintaxis	60 Comprobación	100 Entorno	
30 Construcción Paquetes	70 Datos		
40 Asignación	80 Función		

Tabla 19.3 Ejemplo del cuaderno de registro de defectos.

Estudiante	Estudiante X				Fecha	9/12/96
Profesor	Sr. Z				Programa #	15
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
<input type="text" value="9/12"/>	<input type="text" value="1"/>	<input type="text" value="40"/>	<input type="text" value="codificac."/>	<input type="text" value="CR"/>	<input type="text" value="2"/>	<input type="text"/>
Descripción: omitir declaración de Set_X						
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
<input type="text"/>	<input type="text" value="2"/>	<input type="text" value="80"/>	<input type="text" value="codificac."/>	<input type="text" value="CR"/>	<input type="text" value="8"/>	<input type="text"/>
Descripción: olvidar en el bucle while un desigual						
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
<input type="text"/>	<input type="text" value="3"/>	<input type="text" value="20"/>	<input type="text" value="diseño"/>	<input type="text" value="CR"/>	<input type="text" value="1"/>	<input type="text"/>
Descripción: no poner;						
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Descripción:						
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Descripción:						
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Descripción:						
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Descripción:						
Fecha	Número	Tipo	Introducido	Eliminado	Tiempo de corrección	Defecto corregido
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Descripción:						

tate los pasos. Por otra parte, si piensas que estas pruebas son aún importantes, combina varias de ellas para poder hacerlas más rápidamente. Con cada programa, continúa controlando los datos de defectos y restituye cualquier paso de la revisión que podría haber detectado defectos pasados por alto posteriormente. Cuando los pasos no son efectivos, no dudes en eliminarlos.

19.8

CÁLCULO DEL VERDADERO COSTE DE CALIDAD

Para el PSP, los cálculos simplificados del CDC son adecuados. Cuando trabajas en grandes proyectos de desarrollo, sin embargo, puedes querer utilizar las medidas más precisas de CDC.

Para calcular los verdaderos costes de fallos y valoración, debes dividir los tiempos de revisión, compilación y pruebas entre los componentes respectivos de valoración y fallos. Por ejemplo, podemos denominar el tiempo de compilación en el que no se encuentran defectos como compilación (valoración) o C_V , y al tiempo de corrección de defectos durante la compilación como tiempo de compilación (fallos) o C_F . Así, $C_V + C_F = C$ es el tiempo total de compilación. Para los tiempos de revisión y pruebas, tenemos $R_V + R_F = R$ como tiempo total de revisión y $P_V + P_F = P$ como tiempo total de pruebas.

Utilizando estos parámetros, ahora puedes calcular con precisión la valoración y fallos CDC de la siguiente manera:

$$\text{Valoración CDC} = 100 * (R_V + C_V + P_V) / (\text{tiempo total de desarrollo})$$

$$\text{Fallos CDC} = 100 * (R_F + C_F + P_F) / (\text{tiempo total de desarrollo})$$

El siguiente ejemplo utiliza los datos del Resumen del Plan del Proyecto de la Tabla 19.1 y del Cuaderno de Registro de Defectos de la Tabla 19.3. Primero, calcula los valores de R_V , C_V , P_V , R_F , C_F y P_F de la siguiente manera:

- Para calcular R_V , calcula primero R , a partir del Cuaderno de Registro de Defectos, como la suma de los tiempos de corrección para los defectos eliminados en la revisión de código:
 $R_F = 2 + 8 + 1 = 11$.
- A continuación, calcula R , como $R - R_F = 29 - 11 = 18$.
- Puesto que no se encontraron defectos durante la compilación, todo el tiempo de compilación es tiempo de valoración. Así,
 $C_V = 5$ y $C_F = 0$.
- Puesto que no se encontraron defectos durante las pruebas, todo el tiempo de pruebas es tiempo de valoración. Así, $P_F = 10$ y $P_V = 0$.

Ahora, con estos valores, calculamos los costes de valoración y fallos de la siguiente forma:

$$\begin{aligned}\text{Valoración CDC} &= 100 * (R_V + C_V + P_V) / (\text{tiempo total de desarrollo}) \\ &= 100 * (18 + 5 + 10) / 262 = 100 * 33 / 262 = 12,60\%\end{aligned}$$

$$\begin{aligned}\text{Fallos CDC} &= 100 * (R_F + C_F + P_F) / (\text{tiempo total de desarrollo}) \\ &= 100 * (11 + 0 + 0) / 262 = 100 * 11 / 262 = 4,20\%\end{aligned}$$

Estos valores son algo diferentes a los calculados anteriormente. Se obtiene un valor significativamente más alto de V/F de 3,0 en vez de 1,93. Puesto que los valores de V/F y CDC son más precisos y bastante sensibles a los tiempos de reparación de defectos, no deberías utilizar este método a menos que estés midiendo los tiempos de corrección con un cronómetro. También tendrás como objetivo un valor mayor de V/F ya que ahora un valor de V/F de 2,0 probablemente presentará muchos defectos en las pruebas.

RESUMEN

El coste de calidad (CDC) mide la calidad de un proceso software en términos de 3 componentes: costes de valoración, de fallos y de prevención. El PSP utiliza cálculos simplificados del CDC. El coste de valoración se calcula dividiendo el tiempo de revisión de código por el tiempo total de desarrollo y multiplicando por 100. Este cálculo ignora los tiempos de corrección durante la revisión de código. El coste de los fallos se calcula dividiendo la suma de los tiempos de compilación y pruebas por el tiempo total de desarrollo y multiplicando por 100. Este cálculo ignora los tiempos de compilación y pruebas libre de defectos.

La relación Valoración/Fallos (V/F) mide la calidad del proceso. Se calcula dividiendo el valor CDC de la valoración por el valor del CDC de fallos. La relación V/F indica la cantidad relativa dedicada a encontrar y corregir defectos antes de la primera compilación. El valor de V/F también indica la probabilidad de encontrar defectos del programa en las pruebas. Valores de V/F menores de 1 generalmente indican que se encontrarán muchos defectos en las pruebas, mientras que valores superiores a 2 indican que se encontrarán pocos defectos en las pruebas. Cuando se encuentran pocos defectos en las pruebas, esto indica generalmente que el programa tiene pocos defectos residuales.

Aunque los cálculos simplificados de CDC son ahora adecuados, puedes querer utilizar cálculos más precisos de CDC cuando trabajas en grandes proyectos.

EJERCICIO 19

Como ejercicio, calcula el valor de V/F para todos **los** programas que has escrito en este curso. Determina el número de defectos/KLOC en las pruebas para dichos programas y muestra el valor de V/F frente a **los** defectos/KLOC en las pruebas, en un gráfico similar al mostrado en la Figura 19.1

Entrega copias del Cuaderno de Registro de Tiempos y del Resumen Semanal de Actividades que no hayas entregado previamente. También, entrega acabado el Resumen del Plan de Proyecto del PSP, el Cuaderno de Registro de Defectos y la Lista de Comprobación para la Revisión de Código para cada programa. Incluye los valores Planificados, Reales y Hasta la Fecha de los tiempos de desarrollo, de los defectos introducidos, de los defectos eliminados, del rendimiento y de la relación V/F.

CAPITULO 20

Un compromiso personal con la calidad

El trabajo de calidad es importante para tus jefes, clientes y para ti mismo. Este capítulo discute por qué deberías establecer la calidad como tu máxima prioridad personal.

20.1 LA IMPORTANCIA DE LA CALIDAD

Como ingeniero de software, la calidad de los programas que produzcas tendrá una importancia crítica para tus jefes y clientes. El software se utiliza en muchas aplicaciones y los defectos del software pueden causar perjuicios económicos e incluso algún daño físico. Cuando los programas que escribes son utilizados para trabajos financieros o procesamiento de textos, sus defectos pueden ser molestos y costosos, pero no es probable que nadie sea asesinado o quede mutilado. Cuando tu software forme parte de un sistema de vuelo de aviones, de conducción de coches, de gestión del tráfico aéreo, de funcionamiento de una fábrica o control de plantas nucleares, tus defectos no podrían detectarse y posiblemente tendrían consecuencias peligrosas.

Ha habido personas que han muerto por defectos del software [Leveson]. Aunque no ha habido muchas fatalidades hasta ahora, sin du-

da los números crecerán algo. Esto es porque, a pesar de todos los problemas, el software no se desgasta o deteriora. Es idealmente adecuado para aplicaciones críticas que implican seguridad. Por ejemplo, cuando las actuales plantas generadoras de energía nuclear fueron construidas, los instrumentos de control eran electromecánicos. Era la tecnología más fiable conocida hasta ese momento. Pasados los años, sin embargo, esos primeros instrumentos se han deteriorado gradualmente y han sido reemplazados. Hoy, los sistemas de control más fiables están informatizados. Mientras la primera generación de controladores nucleares era sencilla y utilizaba relativamente poco software, los sistemas de control más sofisticados del futuro implicarán cantidades crecientes de código. Esto significa que la calidad del software será cada vez más importante para aquellos que construyan y gestionen plantas generadoras de energía nuclear. Será también importante para cualquiera que viva cerca.

La misma tendencia es aplicable a cada campo. Los sistemas de control informatizados son tan versátiles, económicos y fiables que los veremos en casi todos los aspectos de nuestras vidas. Es importante que todos los ingenieros de software reconozcan que su trabajo podría tener serios impactos sobre la salud, seguridad y el bienestar de muchas personas.

20.2

EL AUMENTO DEL RIESGO ASOCIADO A LA POCA CALIDAD

Cualquier defecto en una pequeña parte de un gran programa podría, potencialmente, causar serios problemas. Puede parecer improbable que un error tonto en una parte remota de un gran sistema fuese potencialmente desastroso pero, en efecto, estas son las fuentes de problemas más frecuentes. La razón es que los sistemas software son distintos a los sistemas mecánicos, donde los ingenieros pueden construir una protección mecánica. Diseñando adecuados sistemas de seguridad mecánica, los ingenieros pueden hacer que muchas clases de accidentes o fallos sean prácticamente imposibles. Ejemplos comunes aquí son los dispositivos de seguridad utilizados en las prensas para los trabajadores. La prensa tiene acoplado mecánicamente un dispositivo de seguridad, de tal forma que cuando la máquina estampa, **los** operadores no pueden tener nunca las manos dentro.

Con los sistemas software, sin embargo, los sistemas de seguridad generalmente también se hacen con software. Las protecciones de software, son solamente efectivas cuando el software cumple las reglas y normas del sistema. Esto significa que los defectos de software que causan la ruptura de estas normas podrían tener consecuencias impredecibles. Aunque la seguridad del sistema se enfocase desde el nivel del sistema y se utilizasen protecciones hardware y software, no sería siem-

pre práctico. Por lo tanto, para tener sistemas verdaderamente seguros debemos esforzarnos en tener un software libre de defectos.

El diseño de grandes sistemas multiusuario en tiempo real puede ser enormemente complejo. A estas complejas cuestiones de diseño se les presta una gran atención. Aunque las cuestiones difíciles de diseño son normalmente estudiadas, revisadas y probadas con gran cuidado, a los problemas de diseño sencillos se les presta menos atención. Como resultado tenemos que, la causa más común de los problemas con grandes sistemas software, son simples descuidos y tonterías. Aunque muchos de estos errores tontos se detectarán en la integración y pruebas del sistema, algunos seguirán ocultos en el proceso de pruebas y cuando el usuario final utilice el sistema.

El problema es que los ingenieros del software, a menudo, confunden sencillez con facilidad. Consideran que sus constantes errores sencillos serán fáciles de localizar. Se sorprenden, al enterarse, de que dichos errores triviales como: la omisión de un signo de puntuación, la denominación errónea de un parámetro, establecer una condición incorrecta o la terminación errónea de un bucle, podrían pasar las pruebas sin ser detectados y causar serios problemas cuando el usuario utilizase la aplicación. Estos son los tipos de cosas que causan casi todos los problemas, y a las que los suministradores de software dedican millones de dólares a localizar y corregir.

La calidad de **los** grandes programas depende de la calidad de los pequeños programas que lo forman. Así, para producir grandes programas de alta calidad, cada ingeniero del software que desarrolla uno o **más** de estos módulos, debe hacer su trabajo con gran calidad. Esto significa que todos estos ingenieros deben estar personalmente comprometidos con la calidad. Cuando están comprometidos, controlarán y gestionarán **sus** defectos con un cuidado tal, que los pocos defectos que tengan **sus** aplicaciones se encontrarán posteriormente en la integración, en las pruebas del sistema o por los clientes.

Cuando los ingenieros están personalmente comprometidos con la calidad, están orgullos de sus logros. Saben cuánto código han producido y cuántos defectos han encontrado otros en sus productos. Algunos ingenieros me comentan, cuánto tiempo hacía que alguien no había encontrado un defecto en los programas que ellos habían hecho. Cuando los ingenieros del software no están personalmente comprometidos con la calidad, no le dan importancia a esto. Ellos me dirán que en las pruebas encontrarán muchos de los defectos y que el resto que queda, se encontrarán y corregirán fácilmente. No es sorprendente que, cuando los ingenieros piensan de esta forma es poco probable que hagan productos de calidad.

20.3**CÓMO HACER UN COMPROMISO CON LA CALIDAD**

Muchos ingenieros del software ven los defectos como bugs¹ (“gusanos”). Los tratan como algo que arrastran desde alguna parte, pero no es algo de lo que se sientan responsables. Eso es un error. Los defectos son introducidos por los ingenieros y cuando un programa tiene defectos, ¡es defectuoso! El ingeniero que hizo ese programa hizo un programa defectuoso. Cuando los ingenieros están comprometidos con la calidad, prestan atención y cuando prestan atención, son muy cuidadosos con su trabajo. Cuando son muy cuidadosos, producen mejores productos.

El primer paso para producir un software de calidad es decidir que la calidad es importante. El segundo paso es establecer el objetivo de producir programas libres de defectos. Para tener oportunidad de cumplir esta meta, debes medir la calidad de tus programas y dar los pasos que mejoren dicha calidad.

Como seres humanos, todos introducimos defectos ocasionalmente. El reto es gestionar nuestra falibilidad. Esto, sin embargo, es una lucha interminable. No puedes aprender unos pocos trucos de calidad y relajarte. Los productos software del futuro serán más sofisticados y más complejos. Te enfrentarás a problemas que están constantemente cambiando. La cuestión no es si puedes hacer un trabajo libre de defectos, sino, si eres lo bastante cuidadoso para continuar haciéndolo. Esto no solamente hará que tú y tu organización sean más productivos, sino que también tengas más satisfacción por tu trabajo.

20.4**TUS OBJETIVOS PERSONALES**

¿Qué es lo que deseas de la vida? Esta es una gran pregunta que mucha gente tiene problemas al responder. Merece la pena considerar algunas cosas al pensar en la respuesta.

Una forma de conseguir la satisfacción en un trabajo, es tener estatus o poder. Las personas pueden conseguir esto siendo jefes u ocupando cargos en donde desempeñan un servicio importante. El poder y el estatus pueden ser indirectos como ganar gran cantidad de dinero, trabajar para una compañía importante o conducir un coche fabuloso. Estas son todas las partes de “ser” alguien.

Aunque no hay nada malo en lo del estatus, eso es temporal. Puedes ocupar un cargo importante por un tiempo, pero tarde o temprano, tu si-

¹ Es habitual usar el término bug (“gusano”) para denotar **los** errores que hay en un programa. El autor se refiere a esta forma de referenciar los errores.

guiente paso será descender. La pérdida de estatus puede dar lugar a una crisis. Algunas personas quedan sumidas en la tristeza cuando pierden por primera vez un puesto de trabajo importante. Es fácil confundir la importancia de un puesto de trabajo con la importancia personal.

He conocido a directores que quedaron chafados por una degradación. Se habían construido una imagen de sí mismos como personas importantes. Mientras tenían un gran puesto de trabajo, todo el mundo les trataba como personas importantes. En el momento que perdieron el trabajo, sin embargo, eran como cualquier otra persona. Nadie se preocupaba de lo que decían y si dejaron de tener un tratamiento especial. Habían perdido su lujoso despacho y no tenían secretaria. Esto puede suponer un fuerte shock que lleva a algunas personas a tener una crisis nerviosa, ataques al corazón o crisis familiares. **Su recompensa era el estatus y se perdió.**

20.5 LAS RECOMPENSAS DEL LOGRO

Es necesario decidir lo que tú quieras. Piensa por adelantado. Cuando finalmente te jubiles, ¿qué satisfacción te gustaría tener de tu vida? Sugiero que lo que has hecho será más reconfortante que lo que has sido. Si, por ejemplo, planificas trabajar como ingeniero, probablemente tengas el instinto de un constructor. Puede ser que construyas sistemas o componentes. Al final podrías crear métodos o procesos. O puedes tener una inclinación científica y construir teorías o investigar cómo generar el conocimiento fundamental.

De todo lo que construyas, sin embargo, la calidad será la clave. Tendrás poca satisfacción del trabajo descuidado. De algún modo, aunque nadie lo averigüe, sabrás que tu trabajo era descuidado. Esto destruirá tu orgullo en el trabajo y limitará tu satisfacción con la vida. No puedes honestamente decirte a ti mismo que realmente crees en la calidad sin haberla conseguido ni una sola vez. Se pueden poner muchas excusas. Puedes satisfacer a otros con una respuesta oportuna, pero nunca te satisfarás a ti mismo.

Cuando hagas un trabajo de calidad, te sentirás orgulloso. Aunque nadie lo sepa, tu sabes que hiciste un trabajo de primera categoría y estás satisfecho de que hiciste lo mejor. Una cosa sorprendente, es que el trabajo de calidad se reconoce. Puede que pase mucho tiempo, pero tarde o temprano el trabajo de calidad es reconocido. Si sabes esto, tendrás crédito para la calidad de tu trabajo.

Hazte a ti mismo esta pregunta: ¿quiero sentirme orgulloso de lo que hago?. Muchas personas responderían que **sí**. Pero si realmente lo dices en serio, necesitas tener estándares personales y esforzarte en conseguir-

los. Cuando encuentres estos estándares, elévalos y esfuérzate otra vez. Desafíate a ti mismo a hacer un trabajo superior y te sorprenderás de lo que puedes lograr.

REFERENCIA

[Leveson] Leveson, Nancy G. *Safeware, System Safety and Computers*. Reading, MA: Addison-Wesley, 1995.

Índice analítico

A tiempo, en el Cuaderno de Registro de Tiempos 20, 21, 23, 26

A

abstracción 237, 238
definición 237
Ackerman, Frank **A.** 180, 181
actividad 20, 26
clasificación 74
actividades discrecionales 80
actividades exigidas 80
anotación de los datos actuales 212, 213, 214
anotaciones reales, en el Cuaderno de Trabajos
tarea 49, 53
tiempo 49, 53
unidades 49, 53, 70
Ashton Tate 3
asignación de defectos 146

B

Brooks, Frederick P. 104, 112
Brown, Rita Mae 230, 232
bugs 144

C

C (completado) 20
en el Cuaderno de Registro de Tiempos 25
C++ Codificación Estándar 198
C++ Lista de Comprobación para la Revisión
de Código 185

calidad 94, 250
compromiso personal con 163, 279
definición 141
hacer un compromiso con 282
importancia de 4, 279
necesidad del trabajo de calidad 217
proceso 261
producto 247
riesgo asociado a la poca calidad 280
véase también coste de la calidad
Carnegie Mellon University 2
CDC, **véase** coste de la calidad
Chillarege, Ram 146, 161
Codificación, 129
codificación, estándares de 196
C++ 198
código revisión de 166, 175, 229
Ada lista de comprobación 186
antes de la compilación 170, 171, 172
C++ lista de comprobación 185
ejemplo de lista de comprobación 184
guión 171, 189
lista de comprobación 183
lista de comprobación personal 193
objetivos 170
por **qué** la eficiencia 167
utilización de las listas de comprobación
para la revisión de código 187
columnas, en el Resumen Semanal de
Actividades 39
comentarios 20, 26
compilación 128, 129
defectos, datos 172
tiempo 230, 231

- versus defectos post-compilación 272
 versus defectos post-PSP 172
 versus defectos postdesarrollo 173
 versus defectos prueba de unidad 172
 y pruebas, defectos 251,252
 compilación, revisión antes de la 170
 compiladores 164
 objeto 164
 completadas, tareas 24, 26
 comprobación de defectos 146
 compromiso 87
 como contrato 87
 definición 87
 ejemplo 90
 ejemplo en la industria 91
 la gestión de compromisos no conseguidos 92
 lista 95
 personal 87
 responsable 89, 90
 semanal, ejemplo 90
 véase también la gestión de los compromisos
 verdadero 8
 compromisos no conseguidos, gestión 92
 confianza 94
 construcción, paquetes defectos de 146
 contabilización de defectos 153
 coste de la calidad (CDC) 264
 cálculo 265
 cálculo verdadero (CDC) 275
 definición 265
 coste de los fallos 264
 costes de encontrar y corregir los defectos 167, 168
 criterios de entrada, en el Guión del Proceso del PSP 128
 criterios de salida, en el Guión del Proceso del PSP 128
 Cuaderno de Ingeniería 9, 11, 12
 contenidos 15
 Cuaderno de Registro 148, 154
 cabecera 150
 completado 148
 de Tiempos 24
 defecto corregido 150
 descripción 150
 ejemplo 151, 195,274
 eliminar 150
 formulario 149
 instrucciones 150
 introducido 150
 número 150
 propósito 150
 tiempo de corrección 150
 tipo 150
 utilización 154
 diseño 12
 ejemplo 14
 eliminación
 ahorro 221
 economía 217
 eficacia 219
 estrategia 263
 paradoja 262
 problema 218, 250
 rendimiento 250
 tasas, mejora 229
 tiempo 219
 tipo estándar 146
 usos de 12
 Cuaderno de Trabajos 47, 159
 cabecera 53
 con datos de tamaño 68
 ejemplo 51
 formulario 50
 instrucciones 53, 70
 medidas de tamaño 65
 proceso 53
 propósito 53
 sugerencias 49
 Curtis, Bill 236, 246
- ## D
- defecto encontrado, en el Cuaderno de Registro de Defectos 150
 defecto
 datos 147
 análisis 191
 clasificación 192
 ejemplo 206
 utilización 203
 definición 143
 densidad, definición 204
 estimación 205
 introducción
 tasas, reducción 220,230
 y eliminación, ejemplo 222
 y eliminación, experiencia 219
 tasas 204,205,221
 tiempos de corrección 169
 defectos 4, 141, 247
 compilación 172
 versus postdesarrollo 173
 versus defectos post-PSP 173
 versus defectos prueba 172, 251, 252, 271
 contabilización 153

coste de encontrar y corregir 167, 168
datos 172
de datos 146
de documentación 146
de entorno 146
de función 146 diseño, 233
documentación 146
eliminados, plan, en el Resumen del Plan del Proyecto 211
entender 147
entorno 146
estimación 201, 204
formas de encontrar y corregir 164
localización 163
localización con revisiones 169, 170
función 146
importancia de 142
interfaz 146
introducidos, plan, en el Resumen del Plan del Proyecto 211
gestión 221
pasos para encontrar 163
por hora, cálculo 223, 228
por **que** hay que encontrar pronto 167
prueba 234
sistema 146
tiempo para encontrar 178
versus bugs 144
y calidad 142
Defectos/KLOC 201, 204
definición 204
densidad 204
en el Resumen del Plan del Proyecto 210
seguimiento 205
versus años de experiencia 202, 203
definiciones
abstracción 236
calidad 141
codificación estándar 196
coste de la calidad (CDC) 264
coste de los fallos 265
coste de prevención 265
coste de valoración 264
defectos 143
densidad de defecto 204
estándar 196
lista de comprobación 183
proceso 46, 126
producto 46, 126
 proyecto 46, 126
relación valoración/fallos (V/F) 267
rendimiento 219, 251
rendimiento del proceso 225
tarea 46, 126
trabajo 47, 126

descripción
en el Cuaderno de Registro de Defectos 150
en el Cuaderno de Trabajos 53
diagrama lógico 241
Digital Equipment Corporation 169
disciplina, definición 3
diseño 128
defectos 233
causas 237
identificación 235
impacto de 238
naturaleza de 233
véase también defectos
definición 235
proceso 236
representación 239
representación gráfica 240
representaciones en seudocódigo 242
diseño en seudocódigo
ejemplo 242
representación 242

E

eliminación, en el Cuaderno de Registro de Defectos 150
Embry-Riddle, Universidad Aeronáutica (UAER) 108
errores, definición 143
estimación del tamaño del programa 62
estimado, tiempo
en el Cuaderno de Trabajos 47, 53, 70
exactitud de las estimaciones 10, 122

F

FAA (Federal Aviation Administration) 2
Fagan, Michael 176, 180
formularios, por qué utilizarlos 18
Fuerzas aéreas de los EE.UU. 183

G

Gantt, Diagrama 100, 101
ejemplo 102, 106
registro del estado en 107
registro del progreso en 106
gestionar compromisos 87
cómo 95
consecuencias de no 94
importancia de 93
véase también compromiso
Gilb, Tom 176, 180
Gries, David 244, 246

H

Hasta la Fecha
 % Hasta la Fecha
 véase también Resumen del Plan del Proyecto, % Hasta la Fecha
 ejemplo del cálculo 135, 136
 en planificación de defectos 157
 ejemplo del cálculo 135, 136
 máx, en el Cuaderno de Trabajos 47, 48, 49, 53
 mín, en el Cuaderno de Trabajos 47, 48, 49, 53
 tiempo, en el Cuaderno de Trabajos 47, 48, 49, 53
 tiempo, en el Resumen Semanal de Actividades 40
 unidades, en el Cuaderno de Trabajos 47, 48, 49, 53
 véase también Resumen del Plan del Proyecto, Hasta la Fecha
 velocidad, en el Cuaderno de Trabajos 47, 48, 49, 53
 Humphrey Watts S. 112, 113, 123, 126, 139, 144, 146, 161, 176, 180, 202, 215, 232, 250, 259

I

IBM (International Business Machines Corporation) 43, 146, 238
 defectos informados por el cliente 165
 importancia de la calidad 4, 279
 inspecciones 174, 176
 interfaz, defectos 146
 interrupciones, en el Cuaderno de Registro de Tiempos 19, 20
 gestión 22
 tiempo 26
 introducido, en el Cuaderno de Registro de Defectos 150

L

las recompensas del logro, 283
 Leveson, Nancy G. 2, 8, 279, 284
 líneas de código (LOC) 60, 62
 contabilización 60
 lista de comprobación 183
 definición 183
 mejora 194
 personal 188, 196
 por qué ayudan 183
 véase también Lista de Comprobación para la Revisión de Código

Lista de Comprobación para la Revisión de Código en Ada 185, 193
 LOC nuevas y cambiadas 1 16
 LOC/Hora, en el Resumen del Plan del Proyecto 116

M

Marina de los Estados Unidos 4
 medida
 definición 5
 proceso de 261
 mejora 4, 5
 véase también mejora del proceso
 métodos formales 244
 Microsoft NT 219
 minutos/LOC 63
 en el Resumen del Plan del Proyecto 1 16
 Motorola 218

N

NASA (National Aeronautics and Space Administration) 184
 necesidad del trabajo de calidad 217
 número de semanas, en el Resumen Semanal de Actividades 40
 número, en el Cuaderno de Registro de Defectos 150

O

O'Neill, Don 178, 180
 objetivos, personales 282

P

Pareto, distribución 194
 planes
 definición 47
 exactitud 10
 planificación 127, 128
 beneficio 10
 habilidad 10
 véase también planificación del período, planificación del producto
 planificación del período 29
 importancia de 31
 relación con la planificación del producto 30
 por qué ayuda la lista de comprobación 183
 por qué contabilizar defectos 153, 154
 por qué encontrar pronto los defectos 167
 postmortem 128, 129

- prevención, costes de 265
priorizar el tiempo 79
véase también tiempo
Proceso Software Personal (PSP) 2, 3, 7, 126, 146, 155, 163, 218, 222, 240, 248, 257
flujo de 127
rendimiento 263
proceso
calidad 261
definición 46, 126
en el Cuaderno de Trabajos 53
Guion 127, 156, 175, 190, 208
criterios de entrada 128
propósito 128
medida 261
pasos para la mejora 5
por qué utilizarlo 125
rendimiento 225
productividad 59
producto
calidad 247
definición 46
planes
contenidos 45
definición 45
por qué son útiles 44
planificación 29, 43, 113
importancia de 31
necesidad 43
proceso 57
relación entre planificación del periodo 30
tamaño 57
utilización de tiempos y medias 54
programa
funciones 64
tamaño 59, 1 17
ejemplo de estimación 666, 118
estimación 62
formulario de estimación 65
medición 58
programaciones 99
hacer 101
gestión 99
necesidad 99
pasos para hacer 101, 103
sugerencias de seguimiento 106, 108
prototipado 258
proyecto
definición 46
planes, definición 1 13
planes, necesidad para 113
Resumen del Plan 114, 115, 130, 157, 174, 176, 209, 224, 266
cabecera 121, 131
defectos eliminados 211
defectos/hora, cálculo 223
defectos introducidos 211
defectos/KLOC 210
ejemplo 116, 132, 133, 134, 160, 179, 207, 212
en negrita 133
formulario 114, 130
Hasta la Fecha 131
cálculo, ejemplo 135, 136
instrucciones 121, 131, 158, 177, 210, 226, 268
LOC/Hora 121, 131
cálculo 116
minutos/LOC 121, 131
cálculo 116
propósito 121, 131
rendimiento, cálculo 225, 228, 229
sección resumen 116
sombreado 130
tamaño programa (LOC) 121, 131
tiempo en fase 120, 121, 131
seguimiento 105
seguimiento de problemas 106, 108
prueba 128, 129
defectos 234
categorías 234
datos 172
introducida por fases 234
véase también defectos, prueba
versus V/F 270, 271
horas versus V/F 272, 273
tiempo 230
pruebas 248
filtro de 249
sistemas complejos 249
PSP, *véase* Proceso Software Personal
puntos de control 99, 104
ejemplos 104
noadecuados 104
sugerencias 104
y fases 129

R

- Ragland, Bryce 178, 180
recogida de datos de tiempo, por actividad 74
recompensas del logro 283
registro de tiempo 11
ideas para 25
Relación Valoración/Fallos 267
cálculo 267, 268
definición 267
utilización 271, 272, 273

rendimiento 219, 250, 263
 beneficios del 100% de rendimiento 256
 cálculo 225, 251, 268
 ejemplo 253
 estimación del rendimiento definitivo 254
 experiencia 257
 fórmula 251
 mejora 254, 255
 personal 263
 valores 253
 representación gráfica del diseño 240, 241
 respeto 94
 reutilizar 117
 revisiones en parejas 174
 Russell, Glen W. 178, 180

S

seguimiento del tiempo 17
 semanal
 ejemplo de compromisos 90
 estimación de la actividad 79
 ejemplo 80
 Resumen Actividades 29, 31, 74
 cabecera 39
 ejemplo 34, 76
 formulario 32
 instrucciones 39, 40
 propósito 39
 utilización 38
 resumen del tiempo 82
 ejemplo 91, 92
 tiempo programado y estimado, ejemplo 83
 totales, en el Resumen Semanal de Actividades 39
 Shooman, M. L. 178, 181
 símbolos de los diagramas 241
 software
 cronómetro 22, 276
 estándar, definición 196
 fin, en el Cuaderno de Registro de Tiempos 26
 ingeniería
 calidad, véase calidad
 hechos de la vida 95
 medidas proceso 4
 problemas que causa 11
 trabajo 1
 inicio, en el Cuaderno de Registro de Tiempos 26
 proceso de desarrollo 125
 sintaxis, defectos 145
 sistema, defectos 146

T

tamaño 59
 estimación, para programas grandes 64
 máximo 53
 medición 58
 medidas
 en el Cuaderno de Trabajos 65
 objetivos 62
 precauciones sobre 58
 otras medidas de tamaño 62
 véase también tamaño del producto, tamaño del programa
 tarea
 definición 46
 en el Resumen Semanal de Actividades 39
 tareas, completadas 24, 26
 tasas de revisión, mejora 273
 tiempo medio 38
 véase también tiempo
 tiempo variable 79, 80
 sugerencias para gestionar 84
 tiempo
 Cuaderno de Registro 17, 20, 29, 81
 C (completado) 24
 cabecera 26
 completar 20
 con datos pequeños 67
 ejemplo 23, 36, 52
 formulario 21
 ideas 25
 instrucciones 20, 26
 mantener en el Cuaderno de Ingeniería 25
 U (unidades) 24
 datos 11
 distribución 74
 encontrar más tiempo 77
 estimación 77
 ejemplo 75
 gestión 82
 hacer 75
 gestión 9
 elementos de 73
 objetivo 85
 pasos en 11
 reglas básicas para 77
 ideas para registrar tu tiempo 25
 máximo 37
 media 35
 mínimo 37, 53
 períodos 19
 por fase, en Resumen del Plan del Proyecto 120, 121, 131
 por qué controlar 17
 priorización 79

registro **18**
resumen 33
seguimiento 17
tiempos de corrección 81
tiempos incluida la última semana, en el Resumen Semanal de Actividades, 40
tiempos y medias del período 35
cálculo 35
tipo 146
totales, en el Resumen Semanal de Actividades 39
trabajo
definición 46
número, en el Cuaderno de Trabajos 53

U

U (unidades) 20, 26
UAER (Universidad Aeronáutica de Embry-Riddle) 108

utilización de una lista de comprobación para la revisión de código 187

V

V/F, véase Relación Valoración/Fallos
valor ganado 108, **110**
ejemplo 109, 111
estimación 110
seguimiento 108
van Genuchten, Michael 178, **181**

W

Weller, E. F. 178, **181**

Z

Zachary, G. Pascal 219, 232

MATERIAL DE APOYO

En la Introducción al Proceso Software Personal, Watts S. Humphrey proporciona una serie de ejercicios para demostrar el proceso software personal y su utilización.

El material de apoyo de la página web está disponible para ayudar a los estudiantes y lectores a hacer sus ejercicios, los análisis requeridos y los informes. El material contiene kits para hacer los ejercicios indicados en el texto, incluyendo copias de las tablas del libro, y hojas de cálculo para hacer el análisis de datos de los ejercicios. Los lectores encontrarán el material como un complemento práctico del libro. Puede obtenerse en la página web de la editorial, cuya dirección es:

(1) <http://www.librosite.net/humphrey>

Más información para obtener una información actualizada de este libro, sus apéndices y de otros trabajos de interés para profesores de ingeniería del software y profesionales, se puede acceder a la Última información sobre los libros de esta materia de la editorial en la siguiente dirección de Internet (www.pearsoneducation.com).

