Capítulo 1 Qué significa probar software

Introducción

La prueba del software es un proceso que se realiza por diversos motivos, concientemente o de manera casual, pero que se reduce a unos cuantos pasos: se ejecuta el programa (o parte del mismo) en ciertas condiciones, aplicando un conjunto de valores a sus entradas y se observa su respuesta; si esta resulta la esperada o al menos resulta aceptable, entonces se repite con otros valores o en otras condiciones. En cierto momento se da por concluido el proceso y se toma una decisión: aceptar el producto como razonablemente bueno o rechazarlo, regresándolo a revisión.

Este proceso puede realizarse de manera sencilla e informal o de modo muy riguroso y elaborado, pero siempre bajo el mismo principio, que es de tipo experimental.

En su sencillez, el proceso descrito deja una serie de interrogantes:

- ¿por qué lo hacemos?
- ¿para qué sirve?
- ¿cómo determinar las condiciones y entradas adecuadas?
- ¿cómo saber que la respuesta obtenida es correcta?
- ¿cuántas veces debe repetirse o cómo saber cuándo parar? y
- ¿debe aceptarse un producto después de cierto número de intentos?

En este capítulo se detallará y definirá el proceso descrito y se buscará respuesta a los interrogantes.

1.1 Las pruebas en el desarrollo de software tradicional

Todo aquel que ha desarrollado software, ya sea profesionalmente o por entretenimiento, se ha enfrentado con la necesidad de determinar si un programa o parte de él es correcta o no. Usualmente hay dos actividades asociadas con esto: las revisiones llamadas “de escritorio” y las pruebas realizadas al ejecutar el software.

Tanto la revisión de escritorio como las pruebas ejecutando el software utilizan un mismo procedimiento: se asignan valores concretos a las entradas que pueden afectar al software y se observan los resultados que se obtienen. De acuerdo a lo que se esperaba del software, se le da por bueno o se busca qué está mal. El proceso se muestra en la Figura 1.

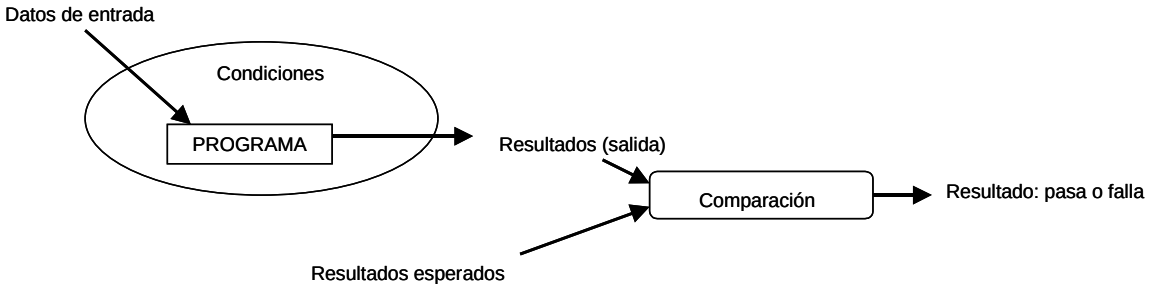

Figura 1 Proceso básico de prueba

A veces no se sabe exactamente cuál debe ser el resultado, pero se tiene una idea acerca de los posibles valores aceptables o se consulta a un experto.

El proceso descrito busca, de alguna manera, asegurar que el programa cumple lo que se espera de él.

Hay otro tipo de pruebas, que realizan los jefes de los programadores, los profesores o personas ajenas que desean verificar la fortaleza del software o aún por simple aburrimiento. En este caso se utiliza el mismo proceso pero con una intención diferente: hallar problemas que hagan fallar al software, pensando que si no es posible, entonces puede ser de buena calidad. Para este enfoque se buscan valores riesgosos o que la experiencia marca como probables causantes de una falla, como valores extremos y valores inesperados.

Otro uso informal de las pruebas ocurre cuando se está depurando un programa y se desea localizar la causa de un problema. Usualmente se dan valores que se cree provocarán la falla y se observan resultados intermedios hasta aislar el segmento de código defectuoso.

En un sentido más amplio, en Informática se emplea la palabra “prueba” cuando se está conociendo un software: “voy a probar el paquete X”. Esto se hace cuando se adquirió una pieza de software o se está evaluando la conveniencia de hacerlo. En este caso se realiza también, de manera informal usualmente, el proceso de la Figura 1.

1.1.1 Ciclo de prueba

Así pues, cuando se habla de “probar software”, se realizan las siguientes acciones:

1) Se forma una idea del tipo de resultado aceptable para ciertos valores de entrada
2) Se ejecuta (o revisa manualmente) el software, ingresando valores específicos en condiciones específicas
3) Se observan los resultados obtenidos
4) Se comparan los resultados con la imagen del resultado que se tenía al inicio

Los pasos 2, 3 y 4 se repiten hasta convencerse de que el software cumple su propósito, se cansa de hacer pruebas el probador o ingeniero de software o se produce una falla que obliga a averiguar la causa. En adelante, cuando se hable de prueba de software se tratará del proceso descrito.

Cada aplicación del proceso incluido en los pasos 2, 3 y 4 se denominará **ciclo de prueba**. Al conjunto de datos involucrados en un ciclo de prueba se le conoce como **caso de prueba** y está formado por:

a) una lista de condiciones de entrada,
b) una lista de valores específicos de las entradas al software,
c) una lista de resultados observables y, opcionalmente,
d) una lista de condiciones que deben cumplirse al terminar.

Cada caso de prueba tiene dos posibles conclusiones: el resultado es el esperado o no lo es; en el primer caso se dice que el caso “pasa” y en segundo que “falla”. Las fallas del software serán de gran importancia.

Supongamos que se desea probar un programa que recibe una cantidad en kilogramos y regresa el valor en libras, entonces dos casos de prueba serían

Condiciones de entrada	Valor de entrada	Resultado esperado	Condición de salida
__	10	22.026	__
__	3	6.607	__

En los ejemplos de arriba no se incluyeron condiciones por ser un problema muy sencillo; se introducen después.

1.1.2 El proceso de prueba dentro del proceso de desarrollo

El proceso de pruebas descrito anteriormente se puede realizar en diversos momentos y por diferentes causas. Sin embargo, el uso principal se da dentro del proceso de desarrollo de software, que puede realizarse de diversas maneras: en una forma secuencial tradicional o de alguna manera iterativa. En cualquiera de ellas existe la actividad de prueba. La aplicación concreta del proceso de prueba dependerá de la forma específica del proceso de desarrollo. Sin embargo, para cualquiera que sea su forma puede descomponerse como se muestra en la Figura 2.

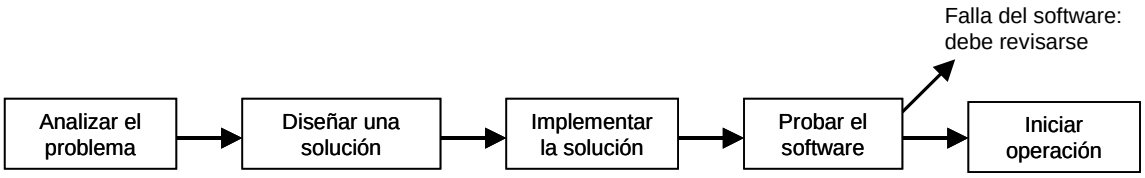

Figura 2 Las pruebas dentro del proceso de desarrollo de software

El esquema de la Figura 2 es muy general. Si se sigue el proceso secuencial tradicional, las pruebas se realizan en las etapas finales del desarrollo, con poca labor de preparación inicial. Si se sigue un método iterativo, las pruebas se van planificando y realizando en cada iteración, a lo largo de todo el proyecto. De cualquier forma, las pruebas se pueden preparar antes de implementar la solución o después. Como se verá más adelante, resulta más provechoso prepararlas antes.

Dentro del proceso descrito las pruebas cumplen dos funciones hasta cierto punto contradictorias: por un lado se busca comprobar que el software o las partes

que se han desarrollado funcionen correctamente; por otro lado, se busca encontrar fallas en su funcionamiento. El primer propósito puede volver complacientes a los desarrolladores, conformándose con unas pocas pruebas que den buenos resultados. El segundo actúa por contradicción: si se realiza un buen esfuerzo por hallar fallas y no se encuentran, entonces el software es bueno. Hay que notar que nunca se puede demostrar la ausencia de fallas, pero sí su presencia.

La comprobación de que el software cumple lo que se espera de él tiene dos variantes, donde ambas son necesarias:

- la comprobación de que el software se realizó correctamente, cumpliendo las especificaciones que se fijaron al inicio, llamada **verificación** y, por otra parte,
- el asegurarse que el software satisface las necesidades del cliente, es decir, le resuelve su problema, a lo que se denomina **validación**.

Más adelante se abundará en este tema.

Hasta aquí se ha hablado de pruebas como de algo natural pero, ¿por qué hacer pruebas? La principal justificación está en la naturaleza humana, siempre propensa a cometer equivocaciones. Este hecho no se puede evitar, ya que todo el software descansa en trabajo humano directo o indirecto (en la realización de herramientas y hardware).

Otro aspecto relacionado con las pruebas y el proceso de desarrollo es el de los niveles de pruebas. En efecto, el software como un todo está formado por subsistemas y estos por partes menores. Las pruebas deben asegurarse que cada parte funcione bien, que las partes se combinen en subsistemas de manera eficaz y que el conjunto total opere correctamente. A grandes rasgos se identifican tres niveles de prueba:

a) **de unidad**, orientada a las partes aisladas;
b) **de integración**, que analizan la interacción de las partes y
c) **de sistema** que analizan al sistema completo.

La Figura 3 muestra un proceso de desarrollo tradicional o una de las iteraciones de uno moderno, mostrando la situación de los diferentes niveles de pruebas.

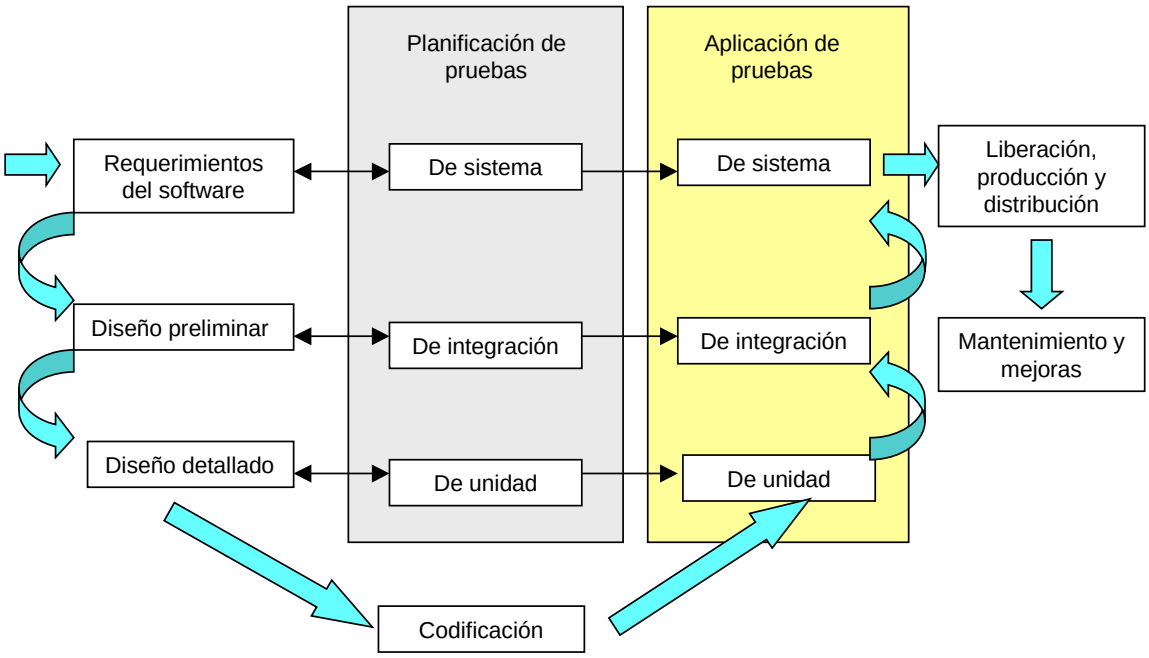

Figura 3 Niveles de pruebas dentro del ciclo de desarrollo (adaptado de A. Davis, 1993)

1.1.3 Registro de resultados

Los casos de prueba que se preparan y sus resultados al ejecutarse, deben quedar registrados para su análisis. A partir de una **falla** ocurrida, se analiza el software buscando su causa, que usualmente es una porción de código o de un modelo que está defectuosa. A este elemento se le llama **defecto**. Los defectos se originan por alguna **equivocación** de alguno de los desarrolladores o aún del propio cliente. Ahora bien, no todas las equivocaciones originan defectos y éstos no siempre ocasionan fallas.

El análisis del número de defectos en relación con la longitud del programa ayudan a estimar qué tan bueno es éste y a saber cuándo se puede dejar de probarlo. Por otra parte, el estudio de los defectos, sus diferentes tipos y los tipos de equivocaciones que los originaron ayudan a entender y mejorar el proceso de desarrollo de software.

1.1.4 Ejemplo

Para ilustrar el proceso de prueba que se describió hasta ahora, considere la siguiente situación: se está desarrollando un sistema de información de una empresa, el cual tiene como requerimiento importante el limitar el acceso a diversas partes del sistema a los empleados y sólo de acuerdo a los permisos con que cuente cada uno. Al analizar el problema se decide que debe existir una clave de identificación para cada empleado y que el sistema debe contar con un módulo que solicite el nombre del empleado y su clave como mecanismo de control.

Antes o después de programar el módulo correspondiente, se analizan las diferentes situaciones que pueden ocurrir: que un empleado real de su clave correcta (el caso más deseable), que un empleado real proporcione una clave

errónea y que un empleado no registrado intente usar el sistema. Para cada situación se prepara un caso de prueba, como los que siguen:

Entrada	Salida esperada
Nombre: González Clave: w12vH8	Mensaje: “Bienvenido Sr. González”
Nombre: González Clave: w1Vh8	Mensaje: “Sr. González, su clave está equivocada”
Nombre: Suárez Clave: w12vH8	Mensaje: “Suárez no es empleado registrado”

Una vez desarrollado el módulo, se ejecuta con cada caso de prueba y se compara el resultado observado con el que se tenía planeado. Suponga que el primer caso produjo el resultado “Bienvenido Sr. González”; entonces ese caso pasa. Ahora, si el segundo produce una salida como “Empleado inexistente”, que no corresponde al resultado esperado, se anotará una falla. Finalmente, si el tercer caso responde “Bienvenido Sr. Suárez”, se debe anotar otra falla. El conteo final será de un caso pasado y dos fallas.

1.2 Generación de Casos de Prueba

En la sección anterior se dio una idea general de lo que es un caso de prueba y cómo se utilizan en la práctica informalmente. Surgen varias preguntas: ¿cuántos casos serán suficientes? ¿cómo generar los menos posibles? ¿qué valores son adecuados?

Conviene precisar lo que se entenderá por caso de prueba:

> Un **caso de prueba** es una especificación precisa de una ejecución de una pieza de software, constituida por: un conjunto de valores de entrada, un conjunto de resultados esperados y opcionalmente, un conjunto de condiciones en que se ejecuta la prueba y un conjunto de condiciones esperadas al terminar dicha prueba.

Los valores de entrada deben ser valores específicos para variables específicas o acciones concretas que originan eventos reconocibles por el software. Ejemplo del primero puede ser nombre: “González”. Un ejemplo de acción sería “oprimir el botón Cancelar”.

Los resultados esperados pueden ser valores específicos de variables, mensajes o cambios en la apariencia del software. Ejemplos del primero sería pago = 1200.00. Ejemplo del segundo: “Operación no permitida”. Del tercero: aparece una ventana para capturar datos de un alumno nuevo.

Los casos de prueba conviene representarlos en un formato como el de la Tabla 1, que contiene los elementos antes descritos.

Tabla 1 Formato para casos de prueba

número	Entradas	Condiciones de entrada	Salida esperada	Condiciones de salida

Las condiciones deben expresarse como proposiciones lógicas que pueden ser falsas o verdaderas. Una condición de entrada puede ser: La base de datos está vacía. Una condición de salida puede ser: Se suspendió la ejecución del programa. Las condiciones de entrada deben ser verdaderas para que se realice el caso de prueba. Las condiciones de salida falsas indican que el caso de prueba falló.

1.2.1 Número de casos de prueba

Considerando la pregunta ¿cuántos casos serán suficientes?, puede analizarse un caso extremo: generar todas las combinaciones posibles de valores que pueda procesar el programa. Si esto fuera posible, se tendría la seguridad de que no hay una combinación que lo haga fallar. Desgraciadamente no es factible, ya que el número de casos crece de manera exponencial respecto al número de variables, debido a que deben considerarse las combinaciones de valores de cada una con las demás. Así pues, habrá que conformarse con un conjunto finito de casos que den cierta tranquilidad.

Para ilustrar el crecimiento del número de casos, suponga que sólo usamos variables booleanas:

- Para una sola, hay dos combinaciones: cierto y falso;
- Para dos variables, se tienen cuatro casos o 2^2: cierto con cierto, cierto con falso, falso con cierto y falso con falso;
- Para n variables tendremos 2^n

Dado que no es factible la prueba exhaustiva, debe buscarse la manera de generar un número suficiente de casos de prueba que ofrezcan cierta seguridad en sus resultados. Para ello deben aplicarse ordenadamente y ser repetibles en diversas circunstancias. En consecuencia, se requiere de un modelo de prueba.

El papel de cualquier modelo es ofrecer una visión simplificada de una fracción de la realidad, de tal manera que pueda razonarse acerca de ella, buscar solución a problemas y ensayar soluciones. En el caso de la prueba de software, el modelo destaca algunos aspectos del software que guían la selección de los casos de prueba, indicando la manera de construirlos y a veces ofreciendo un número mínimo de casos suficientes. Si aún ese número resulta muy grande, pueden usarse métodos estadísticos para obtener una muestra de ellos; en ese caso, la fracción de casos que analizan recibe el nombre de **cobertura**. Idealmente la cobertura del 100% es la mejor, pero a veces basta con un 80%.

1.2.2 Modelos de Prueba

Existen varios modelos para realizar las pruebas, como el combinatorio, el de máquinas de estado, el de caminos de control y el de mutaciones. En los capítulos siguientes se tratarán algunos de éstos.

Desde hace años, cuando se comenzó a probar sistemáticamente, se observó que todo el conjunto de entradas a un programa puede dividirse en varios conjuntos de datos con una propiedad: el programa trata de la misma manera a todos los valores del conjunto: si rechaza a uno, rechaza a todos; si calcula el cuadrado de uno, lo hace con cualquiera de ellos. Sin embargo, el tratamiento de los datos en dos subconjuntos diferentes podía ser distinto; mientras los datos de un conjunto merecían un mensaje de rechazo, los del otro generaban una salida a partir de una fórmula o de un proceso de cálculo. De ahí surgió un método de obtener casos de prueba que corresponde a un modelo combinatorio y que se denomina método de **particiones** o de **dominios**.

Por ejemplo, en un sistema de ventas, sólo se atienden pedidos de 1 a 1000 unidades; si se compran hasta 50 se considera menudeo y se da un precio fijo, como puede ser $5.00 por unidad; a partir de 51 se da precio de mayoreo a $4.50 la unidad. Entonces el dominio de la cantidad pedida será el de los números enteros y se pueden identificar las siguientes particiones con su respectiva acción:

Partición	Acción
Desde - ∞ hasta 0	Error: debe pedir cantidad positiva
Desde 1 hasta 50	Costo =cantidad * 5.00
Desde 51 hasta 1000	Costo = cantidad *4.50
Más de 1000	Error: lo sentimos, no podemos surtir tanto

En principio, basta tomar un ejemplar de cada partición para considerar que se ha probado cualquier valor. Así, tendríamos los siguientes casos de prueba:

Cantidad (entrada)	Costo (salida) o aviso
-4	Error: debe pedir cantidad positiva
5	25.00
70	315.00
2000	Error: lo sentimos, no podemos surtir tanto

Asociado con el mismo modelo existe otro método, nacido de la observación de los programadores: los valores más propensos a causar problemas son aquellos que están en las fronteras de los conjuntos antes descritos. Ésto es muy evidente, por ejemplo, cuando se utilizan métodos de ordenamiento en arreglos, archivos secuenciales o listas ligadas: los casos donde ocurren más fallas son en el primer elemento y en el último. Así pues surge otro

método conocido como de **valores a la frontera**. Las ventajas de éste método son mayores cuando se trabaja con algoritmos numéricos de punto flotante, que son muy sensibles a la precisión de los resultados.

Por ejemplo, para el mismo caso del ejemplo anterior, deben considerarse tres fronteras: de la partición uno a la dos, de la dos a la tres y de ésta a la cuatro. Algunos casos de prueba para estas fronteras son los siguientes:

Frontera	**Cantidad (entrada)**	**Costo (salida) o aviso**
Entre uno y dos	0	Error: debe pedir cantidad positiva
Entre uno y dos	1	5.00
Entre dos y tres	50	250.00
Entre dos y tres	51	229.50
Entre tres y cuatro	1000	4500.00
Entre tres y cuatro	1001	Error: lo sentimos, no podemos surtir tanto

En la práctica se usan mucho los dos métodos anteriores de manera combinada. Estos métodos se describen en el Capítulo dos.

Los modelos de particiones y de valores a la frontera consideran básicamente las entradas al software que se analiza, sin preocuparse de cómo se realizó dicho software. Por eso se les llama de **caja negra**, ya que no se basan en el contenido del código. Sin embargo, no está prohibido observar cómo se programó. Estos métodos también se llaman funcionales, ya que sólo prueban la funcionalidad que se desea realice el software.

Los métodos de caja negra son muy usados para pruebas de sistema y para preparar pruebas antes de codificar. Esta es una técnica muy útil para prevenir defectos y es la base de métodos de desarrollo llamados “**métodos dirigidos por pruebas**” y especialmente el más famoso conocido como Programación extrema.

Los métodos de caja negra son muy útiles, especialmente para probar lo que queremos que haga el software. Sin embargo no son suficientes para asegurar la calidad del software. Como complemento existen métodos que omiten lo que se desea hacer y se concentran en lo que realmente hace el software, analizando su estructura. Estos métodos son conocidos como de **caja blanca** o **estructurales**.

La mayoría de los métodos de caja blanca analizan los caminos que recorre la ejecución de un programa entre las diversas instrucciones. Unos buscan asegurar que todas las instrucciones se ejecuten al menos una vez, otros que

todas las decisiones (if) se ejerciten en sus dos valores. Un método que cubre ambas características es el **método de caminos básicos**, propuesto por McCabe.

A manera de ejemplo, suponga la siguiente implementación en Pascal del ejemplo del costo según cantidad (los números de línea se agregan para la explicación y no son parte del programa):

```
program venta;
const preciobase=5.00; preciomayoreo=4.50;
var cant: integer; costo: real;
 1 function calcula(k:int):real;
 2 begin
 3      if (k<1) then
 4      begin
 5         println('Error: debe ordenar una cantidad positiva');
 6          calcula:=-1;
 7     end
 8      else
 9           if (k<=50) then
10                calcula :=k * preciobase
11           else
12              if (k<1000) then
13                   calcula := k * preciomayoreo
14              else
15              begin
16                  println('Error: no podemos surtir más de mil');
17                calcula := -2;
18             end;
19 end;
…
end.
```

De acuerdo al método de caminos se pueden establecer un grafo como el de la Figura 4

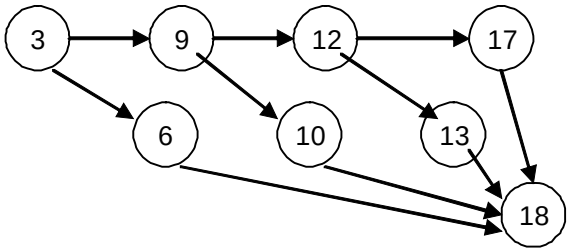

Figura 4 Grafo de control del programa de ejemplo

Se dice que el grafo tiene complejidad ciclomática cuatro y, por ello, tiene cuatro caminos básicos. A partir del grafo se pueden establecer tales caminos:

3 -- 6 – 18

3 – 9 – 10 – 18

3 – 9 – 12 – 13 – 18

3 – 9 -- 12 – 17 – 18

Para cada camino se elige un caso de prueba como pueden ser:

K (entrada)	**Calcula (salida) y aviso**
0	-1; Error: debe ordenar una cantidad positiva
10	50.00
100	450.00
2000	-2; Error: no podemos surtir más de mil

Note que la implementación que se muestra presenta algunas diferencias con el ejemplo inicial, lo cual es de esperar en la práctica. También existe un error de implementación que no sería detectado por estos casos de prueba, pero sí por los de valores a la frontera. Por eso se recomienda usarlos juntos.

El método de caminos se tratará en el capítulo tres.