1 Relación de ejercicios. *Tema 4*

61. Categorizar las siguientes situaciones como: *fallo*, *defecto* o *error*. Para cada uno de ellos, explicar por qué pertenece a esa categoría.
    (a) Un ingeniero de software, trabajando apresuradamente, borra inintencionadamente una línea importante de código fuente
    (b) El 1 de Enero de 2040 el sistema informa que la fecha es 1 Enero de 1940
    (c) No se suministra ninguna documentación de diseño o comentarios en el código fuente para un algoritmo muy complejo
    (d) Un array de tamaño fijo de longitud 10 se utiliza para mantener la lista de cursos de un estudiante durante un semestre. Los requisitos no se pronuncian acerca del número máximo de cursos que un estudiante puede cursar al mismo tiempo.

62. ¿Qué ocurre cuando el siguiente programa `java` es llamado: (a)sin argumentos, (b)con una cadena como argumento, (c) con argumentos enteros 0, 1, 2 y 99?

```
class MiExcepcion extends Exception{
  public MiExcepcion(){ super();}
  public MiExcepcion(String s){ super(s);}
}
class MiOtraExcepcion extends Exception{
  public MiOtraExcepcion(){ super();}
  public MiOtraExcepcion(String s){ super(s);}
}
class MiSubExcepcion extends Exception{
  public MiSubExcepcion(){ super();}
  public MiSubExcepcion(String s){ super(s);}
}
public class throwtest{
   public static void main(String argv[]){
     int i;
     try{
       i= Integer.parseInt(argv[0]);
     }
     catch(ArrayIndexOutOfBoundsException e){
       System.out.println("Debes especificar un argumento");
       return;
     }
     catch(NumberFormatException e){
       System.out.println("Debes especificar un argumento entero");
       return;
     }
     a(i);
   }
public static void a(int i){
  try{
    b(i);
  }
  catch(MiExcepcion e){//Punto 1
    if(e instanceOfMiSubExcepcion)
      System.out.print("MiSubEXcepcion:");
    else
      System.out.print("MiExcepcion:");
    System.out.println(e.getMessage());
    System.out.println("Manejado en el punto 1");
  }
}
public static void b(int i) throws MiExcepcion{
  int result;
  try{
    System.out.print("i="+i);
    resultado= c(i);
    System.out.print("c(i)="+resultado);
  }
  catch(MiOtraExcepcion e){  //Punto 2
```

```
50      System.out.println("MiOtraExcepcion:"+e.getMessage());
51      System.out.println("Manejado en el punto 2");
52    }
53    finally{
54      System.out.print("\n");
55    }
56  public static int c(int i) throws MiExcepcion, MiOtraExcepcion{
57    switch(i){
58      case 0: throw new MiExcepcion("entrada demasiado baja");
59      case 1: throw new MiSubExcepcion("entrada todavia muy baja");
60      case 99: throw new MiOtraExcepcion("entrada muy alta");
61      default: return i*i;
62    }
63  }
64  }
```

63. Examinar las categorías de defectos en el esquema de clasificación de la Hewlett-Packard, mostrado en la figura siguiente ¿Corresponde a una clasificación ortogonal? si no lo fuera, explicar por qué y proponer las maneras de hacerla ortogonal.

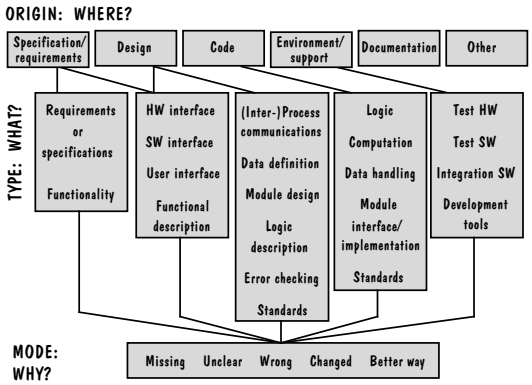


Figura 1: Clasificación de defectos de Hewlett-Packard

64. Sea P un componente de un programa que lee una lista de N registros y un *rango* de condición sobre la clave del registro. Por ejemplo, el rango: 'JONES' .. 'SMITH' producirá como archivo de salida que contiene todos los registros cuyas claves estén comprendidas lexicográficamente entre 'JONES' y 'SMITH'. Los primeros 7 caracteres del registro constituyen la clave. El componente P lee la clave y produce un archivo de salida que contiene únicamente aquellos registros que caen dentro del rango prestablecido. Escribir las condiciones de entrada y salida como afirmaciones para ser utilizadas para verificar si P es correcto. Preparar un diagrama de flujo para mostrar cómo podría ser el flujo lógico de P e identificar los puntos de transformación.
65. Completar la prueba del ejemplo ilustrado en la figura 2. En otras palabras, escribir los enunciados que se corresponden con el diagrama de flujo. Encontrar también los caminos entre la condición de entrada y la de salida.

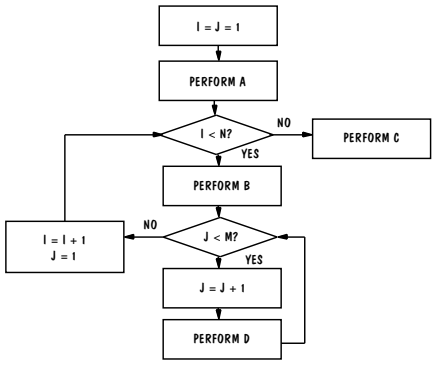


Figura 2: Ejemplo de estructura lógica expresada como diagrama de flujo

66. Suponer que un programa contiene n puntos de decisión, cada uno de los cuales tiene dos ramas ¿Cuántos casos de prueba se necesitan para realizar la prueba de caminos?¿Puede deducirse ese número gracias a la estructura del programa? Dar un ejemplo que confirme y justifique las respuestas.

67. Considerar un diagrama de flujo de un programa como un grafo dirigido en el cual los rombos y cajas del programa se consideran como nodos y las flechas de flujo lógico entre ellos como aristas orientadas del grafo. Por ejemplo el programa de la figura 3 siguiente puede graficarse como se muestra al lado. Probar que la prueba de sentencias de un programa es equivalente a encontrar el conjunto de caminos del grafo que contienen a todos los nodos del grafo. Probar que la prueba de ramificación es equivalente a encontrar el conjunto de caminos cuya unión cubre el conjunto de aristas del grafo. Por último, probar que la prueba de caminos es equivalente a encontrar todos los caminos posibles a través del grafo.

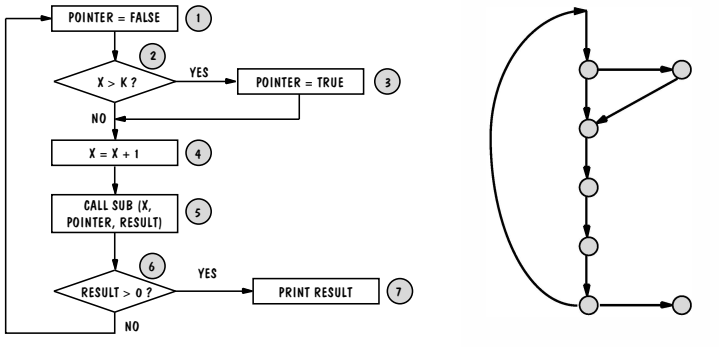


Figura 3: (a) Diagrama de flujo de un programa simple (b) Grafo orientado equivalente

68. *Problema programable:* escribir un programa que acepte como entrada los nodos y aristas de un grafo orientado e imprima como salida todos los posibles caminos a través del grafo ¿Cuál es la principal consideración de diseño para este programa?¿Cómo afecta la complejidad del grafo al algoritmo utilizado?

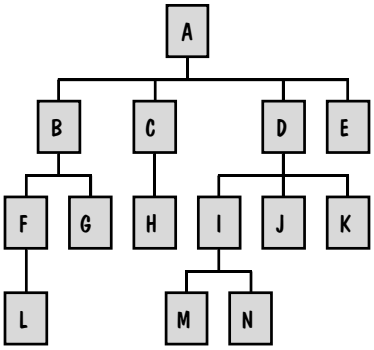


Figura 4: Ejemplo de jerarquía de componentes

69. La figura 4 muestra la jerarquía de componentes de un sistema software. Describir la secuencia de pruebas de integración de los componentes utilizando los diversos enfoques: ascendente, descendente, descendente modificado, *big-bang*, emparedado y emparedado modificado.
70. ¿Cuáles son las explicaciones posibles para el comportamiento de la gráfica presentada en la figura 5 anterior?
71. Un programa se siembra con 25 defectos. Durante la prueba, se detectan 18 defectos, 13 de los cuales son defectos sembrados y 5 son defectos autóctonos, ¿Cuál es la estimación del número de defectos autóctonos que permanecen sin detectar en el programa?
72. Un programador afirma que sus programas están libres de defectos con un nivel de certeza del 95%. El plan de prueba afirma que los programas han de ser probados hasta encontrar la totalidad de los defectos sembrados ¿Con cuántos defectos debe sembrarse el programa antes de la prueba a fin de probar la afirmación anterior? Si por alguna razón el programador no piensa encontrar todos los defectos sembrados, ¿cuántos defectos sembrados requiere la fórmula de Richards en su lugar?

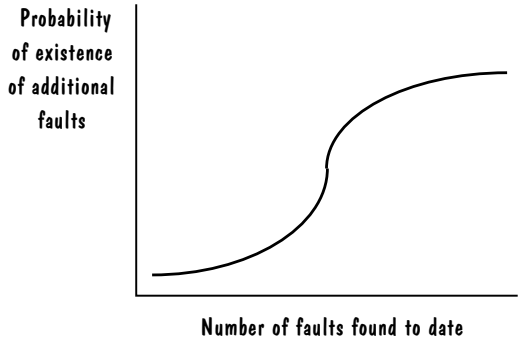


Figura 5: Probabilidad de localizar defectos durante el desarrollo

73. Explicar por qué la gráfica de la figura 5 puede interpretarse para que signifique que, si se encuentran demasiados defectos en el código al momento de la compilación, lo mejor es descartar el código y escribirlo de nuevo.
74. Crearse una tabla de clases de equivalencia para cada uno de los siguientes problemas de única entrada. Ayuda: situar una entrada en una clase de equivalencia distinta si existe incluso una remota posibilidad de que algún algoritmo razonable pudiera tratar la entrada de una manera especial.
    (a) Un número telefónico para ser utilizado por un marcador automático
    (b) El nombre de una persona escrito con caracteres latinos
    (c) Una franja horaria, que puede ser especificada bien nuéricamente dando su diferencia con UTC (GMT) o alfabéticamente utilizando un conjunto de códigos estándar (EST,BST,PDT, etc.)
    (d) La velocidad de un vehículo, que es un entero de 3 dígitos, que podría ser seguido por las unidades: km/h, m/s, o mph (km/h es la unidad por defecto)
    (e) El número de una tarjeta de crédito
    (f) Una frecuencia de radio FM
    (g) Un URL
75. Describir un buen conjunto de equivalencia de clases de test para un formulario que pregunta información personal: apellidos, nombre, fecha de nacimiento, calle, ciudad, país,código postal y teléfono particular.
76. Ampliar la respuesta del ejercicio anterior para incluir valores frontera de clases de equivalencia que hayan de ser probados.
77. Crear las tablas que sean precisas para ayudarnos a probar las condiciones correspondientes con los siguientes requerimientos.
    (a) Los solicitantes de menos de 18 años deben utilizar el formulario A, mientras que los que tengan 18 o más deben utilizar el formulario B. Las personas minusválidas de cualquier edad, excepto aquellos con asistencia social, deben utilizar el formulario C. Las personas con asistencia social deben completar el formulario D, además del A o del B. Los ciudadanos mayores deben completar el formulario E además de otros formularios, salvo que tengan asistencia social o estén viviendo en una residencia. Cualquiera que gane más de 15.000 EUR al año debe rellenar el formulario F además de los otros
    (b) El usuario C envía un mensaje al usuario A. El mensaje debe ser reenviado de A a B si las siguientes condiciones son ciertas: 1) A ha solicitado que sus mensajes sean reenviados a B; 2) B está conectado; y 3) B no ha pedido que se bloqueen los mensajes que le lleguen de A o C. Además si se declara el mensaje como de emergencia, entonces no puede ocurrir ningún bloqueo.

(c) El sistema de navegación debe anunciar a los conductores que tienen que girar hacia una nueva salida dos minutos antes. Se pueden producir excepciones en las siguientes circunstancias. Primera, si un conductor está completando todavía un giro anterior, entonces el aviso del nuevo giro debe ser retrasado hasta que el primer giro termine, pero aún en ese caso se le debe dar un aviso de 15 segundos. Segunda, en un entorno urbano (donde el conductor está circulando bajo un límite de velocidad) encontraría, al menos, un posible giro cada 30 segundos, entonces el sistema avisará al conductor sólo 30 segundos antes que se necesite girar.

78. Java posee una capacidad de clasificación incluida, que se encuentra en las clases `Array` y `Collection`. Probar experimentalmente si estas clases contienen o no algoritmos estables y eficientes.
79. Describir el tipo de defectos numéricos presentes cuando el siguiente código es programado en un programa. Suponiendo que no se conocía la implementación, explicar cómo haríamos para intentar detectar tales defectos:
    (a) `double x, y; ... if(x/3.0==y){...}`
    (b) `int activosCorporativosTotales; //En EUR`
    (c) `short precioGasolina; // En USD decimas de centavo por galon`
    (d) Administramos una página Web que implementa micro–pagos; le factura a los usuarios 1/3 de céntimo por cada click en la página. Acumulamos la factura de cada cliente en un entero donde cada unidad representa 1/10 de céntimo. Sin embargo, registramos el dinero ganado por cada página con un valor en coma flotante.
80. Si estuviéramos diseñando los siguientes tipos de software, a qué pruebas de *stress* y situaciones excepcionales someterías el sistema:
    (a) Un navegador Web nuevo
    (b) Un simulador de vuelo
    (c) Un sistema de chat simple
    (d) Un sistema de navegación
81. Escribir casos de test completos para cada una de las situaciones de prueba listadas en el ejercicio 67
82. Expandir la siguiente tabla en un plan de pruebas completo:

Etapa de vuelo	Visibilidad	Tren de aterrizaje	Resultado requerido
2-3 min. del despegue	$\geq$ 1000 ft	Desplegado	Sin alarma
2-3 min. antes aterrizar	$\leq$ 1000 ft	Desplegado	Sin alarma
2-3 min. del despegue	$\geq$ 1000 ft	No desplegado	Sin alarma
2-3 min. antes aterrizar	$\geq$ 1000 ft	No desplegado	Sin alarma
2-3 min. del despegue	$>$ 1000 ft	Desplegado	Sin alarma
2-3 min. antes aterrizar	$>$ 1000 ft	Desplegado	Sin alarma
2-3 min. del despegue	$>$ 1000 ft	No desplegado	Alarma!
2-3 min. antes aterrizar	$>$ 1000 ft	No desplegado	Alarma!

83. Discutir cómo la *prueba de integración* podría realizarse en un sistema de chat simple ¿Qué *stubs* o *drivers* podrían escribirse para permitir realizar una prueba independiente de las capas de componentes?
84. ¿Por qué puede ser muy complicado realizar pruebas unitarias a un módulo altamente acoplado?

85. ¿Es siempre posible desarrollar una estrategia para probar software que utiliza la secuencia de pasos de prueba descrita más abajo?¿Qué posible complicaciones podrían aparecer en los sistemas empotrados?

    (a) Pruebas unitarias

    (b) Pruebas de integración

    (c) Pruebas de validación

    (d) Pruebas del sistema

86. ¿Cómo puede la planificación de un proyecto afectar a la prueba de integración?
87. ¿Son posibles o incluso deseables las pruebas unitarias en cualquier circunstancia? Proporcionar ejemplos para justificar la respuesta.
88. ¿Quién debería realizar la prueba de validación: el desarrollador o el usuario del software? Justificar la respuesta.
89. Discutir las diferencias entre las pruebas de un sistema crítico para el negocio, un sistema de seguridad crítica y un sistema cuya falla no afecta seriamente a las vidas, la salud o los negocios.
90. Dar un ejemplo de un sistema orientado a objetos donde los problemas de sincronización requieran una prueba cuidadosa.
91. Si un equipo de prueba independiente realiza una prueba de integración, y un defecto crítico permanece en el código después de completar la prueba ¿quién es legal y éticamente responsable por el daño que ha causado el defecto?
92. Suponer que se está construyendo un sistema para la preparación de impuestos que tiene tres componentes:

    (1) Crea formularios sobre pantalla que permiten que el usuario teclee su nombre, dirección, número de identificación impositiva e información financiera

    (2) Utiliza tablas de impuestos y la información ingresada por el contribuyente para calcular el monto a pagar para el año actual

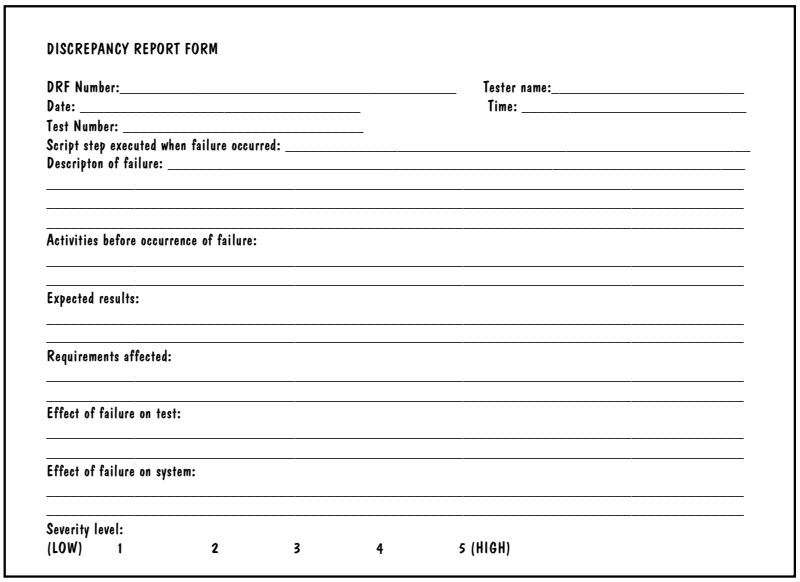
    (3) Utiliza información del domicilio del contribuyente para imprimir formularios para el pago de las diferentes contribuciones (nacional,provincial,municipal), incluyendo el monto a pagar.

    Presentar la estrategia que podría utilizarse para probar este sistema y delinear los casos de prueba en un plan de pruebas.

93. Considerar el desarrollo de un ensamblador en dos pasos. Perfilar sus funciones y describir cómo se podría probar para que cada función se probara completamente antes de pasar a examinar la próxima función. Sugerir un plan de construcción para el desarrollo y explicar cómo se pueden diseñar juntos el plan de construcción y el de pruebas.
94. La certificación es una aprobación otorgada por una fuente externa que garantiza la exactitud de un sistema. Se concede comparando el sistema con un estándar predefinido de rendimiento. Por ejemplo, el Departamento de Defensa (DoD) Norteamericano certifica un compilador del lenguaje Ada después de probarlo contra una extensa lista de especificaciones funcionales. En la terminología introducida en este tema, tal prueba, de qué tipo sería:
    - Rendimiento
    - Aceptación
    - Instalación

    Justificar en cada uno de los casos anteriores.

95. Cuando se desarrolla un plan de construcción, es necesario tener en cuenta los recursos disponibles tanto para los desarrolladores y los clientes, incluidos tiempo, personal y dinero. Presentar ejemplos de restricciones de recursos que pueden afectar el número de construcciones de prueba (*builds*) definidos para el desarrollo del sistema. Explicar cómo estas restricciones afectan el plan de construcción.



DISCREPANCY REPORT FORM

DRF Number:_____ Tester name:_____
Date: _____ Time: _____
Test Number: _____
Script step executed when failure occurred: _____
Descripton of failure: _____

Activities before occurrence of failure:

Expected results:

Requirements affected:

Effect of failure on test:

Effect of failure on system:

Severity level:
(LOW) 1 2 3 4 5 (HIGH)

Figura 6: Formulario para informar de discrepancia

96. Suponer que la calculadora de un matemático posee una función que calcula la pendiente y la intersección de una línea. El requerimiento en el documento de definición dice: "'La calculadora aceptará como entrada una ecuación de la forma Ax+By+C = 0 e imprimirá como salida la pendiente y la intersección". La implementación del sistema para este requerimiento es la función LINE(A,B,C), donde A y B son los coeficientes de x y de y. La constante de la ecuación es C. El resultado es una salida impresa de D y E, donde D es la pendiente y E la intersección. Escribir este requerimiento como un conjunto de causas–efectos y dibujar el gráfico correspondiente.
97. Los requerimientos han de ser comprobables, es decir, que se puedan *probar*. Explicar por qué la facilidad de prueba es esencial para la prueba de rendimiento. Utilizar ejemplos para justificar la explicación.
98. ¿Qué tipos de pruebas de rendimiento podrían requerirse para un sistema procesador de palabras?¿Un sistema de nómina de pagos?¿Un sistema de cajero bancario automatizado?¿Un sistema de supervisión de calidad del agua?¿Un sistema de control para una planta de energía?
99. Un sistema de control de tráfico aéreo puede diseñarse para que sirva a un único usuario o a muchos. Explicar de qué manera un sistema como éste puede tener una variedad de configuraciones y describir cómo se podría diseñar un conjunto de pruebas de configuración?
100. Un sistema de navegación está a punto de ser instalado en un avión. ¿Qué problemas deben ser considerados al diseñar la prueba de instalación?
101. Dar un ejemplo para demostrar que, sin la utilización de un simulador del dispositivo, la prueba a veces es imposible de realizar. Dar otro ejemplo para demostrar la necesidad de un simulador del sistema.
102. Hacer un comentario sobre el formulario de informe de discrepancia presentado en la figura 6, a la luz de las preguntas que es necesario contestar acerca de la falla a partir de la lectura del formulario.
103. Un sistema de salarios está diseñado de manera que hay un registro de información de empleado por cada persona que trabaja para la compañıa. Una vez por semana, el registro del empleado se pone al día con el número de horas trabajadas por el empleado durante la semana. Cada dos semanas se imprimen los resúmenes, para desplegar el número de horas trabajadas desde el comienzo del año fiscal. Una vez al mes, la retribución mensual de cada empleado durante el mes se transfiere electrónicamente a su cuenta del banco. Para cada uno de los tipos de pruebas de rendimiento descritos en este capítulo, indicar si corresponde aplicarse a este sistema.

104. “Segarra Alpargatas" ha comisionado a un equipo de la ETSIIT de Granada para que desarrolle un sistema basado en computadora para probar la resistencia de su línea completa de calzado de goma. Segarra tiene nueve fábricas en varias ubicaciones alrededor del mundo y cada sistema se configurará según el tamaño de la fábrica. Explique por qué Segarra y el equipo de la ETSIIT deben dirigir la prueba de instalación aun cuando la prueba de aceptación se haya completado satisfactoriamente.
105. Escribir un guión de prueba para probar la función LINE descrita en el ejercicio 96.
106. Se ha propuesto una medida de la confiabilidad en términos del tiempo medio hasta la falla, de la disponibilidad en términos del tiempo medio entre fallas y de la facilidad de mantenimiento en términos de tiempo medio para reparar ¿Esta medidas son consistentes con las definiciones presentadas? Es decir, si se definen *confiabilidad*, *disponibilidad* y *facilidad* de mantenimiento como probabilidades, ¿se obtentrán los mismos números que si se usan las métricas? Si no es así, ¿un método puede transformarse en el otro, o existen diferencias básicas irreconciliables?
107. Un sistema de seguridad crítica falla y se pierden varias vidas. Cuando se investiga la causa de la falla, la comisión a cargo descubre que el plan de prueba desestimó la consideración del caso de prueba que ha causado la falla del sistema ¿Quién es el responsable de las muertes: los verificadores por no notar el caso omitido?¿Los planificadores de la prueba por no escribir un plan de prueba completo?¿Los gerentes por no haber verificado el plan de la prueba?¿El cliente por no haber hecho una prueba de aceptación completa?
108. Si los requerimientos de un sistema de confiabilidad muy elevada significan que la confiabilidad nunca puede verificarse, a pesar de eso, ¿el sistema debe usarse?
109. A veces, los clientes contratan una organización independiente (separada de la organización de desarrollo) para realizar una verificación y validación independiente (V&V). El personal de V&V examina todos los aspectos del desarrollo, incluso el proceso y producto para asegurar la calidad del producto final. Si se emplea un equipo de V&V independiente y el sistema todavía experimenta una falla catastrófica, ¿quién debe considerarse responsable?: ¿los gerentes, el equipo de V&V, los diseñadores, los codificadores o los verificadores?
110. Se definen dos funciones:
    (a) Función de distribución:$F(t) = \int_{t_1}^{t_2} f(t)\,\mathrm{d}t$
    (b) Función de confiabilidad: $C(t) = 1 - F_i(t)$
    - Función de densidad de probabilidad: $f(t)$, describe la comprensión de cuándo es probable que el software falle
    - La función de confiabilidad $C(t)$ se define como la probabilidad de que el software funcione correctamente hasta el instante t.

    Si la confiabilidad de un sistema mejora cuando se lo prueba y arregla, ¿qué les sucede a los gráficos de dichas funciones?
111. Se describen dos versiones de software VxWorks, uno para un chip PowerPC y otro para un chip R6000. Explicar los problemas de gestión de la configuración relacionados con la construccón de un sistema para dos chips diferentes ¿Podría la estrategia de gestión de configuración haber ayudado a que el vendedor

“portara" la versión del PowerPC al R6000?

Había varios sistemas operativos disponibles para el microprocesador R6000 que llevaría el dispositivo de aterrizaje Pathfinder que la NASA colocó en el Sojourner para Marte. La NASA seleccionó VxWorks de WindRiver Systems. Cuando llegó, el VxWorks estaba probado y listo para PowerPC, pero la versión para el R6000 todavía no estaba lista. Wind River Systems convirtió la versión para PowerPC del sistema VxWorks al R6000, en lugar de utilizar un desarrollo nativo, sacando ventaja de la supuesta portabilidad del lenguaje C. Por lo tanto, el software del Pathfinder estaba construido con una versión de prueba *beta* de su sistema operativo, en lugar de estar montado sobre un sistema robusto y completamente probado.
En Julio de 1997, debido a las fallas relacionadas con la administración de pilas y punteros durante los procesos de conmutación, el Pathfinder se reinicializó e interrumpió su trabajo durante algunos periodos de tiempo en su misión a Marte.

112. Un oráculo de prueba es una persona o máquina hipotética que puede decir cuándo los resultados de la prueba reales son iguales que los resultados esperados. Explicar la necesidad de inclusión de un oráculo de pruebas en el desarrollo de la teoría de la prueba.