# 1 Relación de ejercicios. *Tema 5*

113. Categorizar los siguientes sistemas como sistemas–S, –P, o –E. Para cada uno de ellos, explicar por qué pertenece a esa categoría. Identificar aquellos aspectos del sistema que pueden cambiar.
   (a) Un sistema de control de tráfico aéreo.
   (b) Un sistema operativo para un microordenador.
   (c) Un sistema de aceleración de operaciones en coma flotante.
   (d) Un sistema de gestión para una base de datos.
   (e) Un sistema para encontrar los factores primos de un número.
   (f) Un sistema para encontrar el primer número primo mayor que un número dado.
114. Explicar por qué un grado alto de acoplamiento entre los componentes de un sistema puede hacer que su mantenimiento resulte muy difícil.
115. Explicar por qué el éxito de un sistema depende en gran medida de la calidad de la documentación generada durante el desarrollo del sistema.
116. Considerar el proyecto de prácticas que se ha desarrollado durante la segunda práctica de esta asignatura, comenzó siendo un proyecto de desarrollo de tamaño mediano y se ha ido reforzando durante el desarrollo de curso (diseño complejo de interfaces, pruebas unitarias, etc.) continuamente hasta que el resultado final sea un proyecto completo. Repasando las notas que se hayan tomando durante el desarrollo del proyecto de prácticas, se pide contestar a las siguientes preguntas:
   (a) ¿Cuánto tiempo se ha necesitado para definir y entender el problema?
   (b) ¿Cuánto tiempo llevó el implementar todo el código?

   Comparar las categorías de la Tabla siguiente con las estimaciones de tiempo para el citado proyecto, y comentar si las diferencias son buenas o malas.

| | Muy bajo | Bajo | Nominal | Alto | Muy Alto |
|---|---|---|---|---|---|
| Estructura | Cohesión muy baja acoplamiento elevado, código “spaghetti" | Cohesión moderada baja, acoplamiento elevado | Razonablemente bien estructurado; algunas áreas débiles | Cohesión alta acoplamiento bajo | Sólida modularidad ocultamiento de información en datos y estructuras de control |
| Claridad de la aplicación | Ninguna concordancia entre el programa y la visión del mundo de la aplicación | Alguna correlación entre el programa y la aplicación | Moderada correlación entre el programa y la aplicación | Buena correlación entre el programa y la aplicación | Clara concordancia entre el programa y la visión del mundo de la aplicación |
| Capacidad de autodescripción | Código oscuro; falta documentación, oscura u obsoleta | Algún comentario de código, encabezados; alguna documentación útil | Moderado nivel de comentarios de código, encabezados y documentación | Buenos comentarios código y encabezados; documentación útil áreas débiles | Autodescriptivo; documentación actualizada, bien organizada, con justificación diseño |
| Incremento de la comprensión del software | 50% | 40% | 30% | 20% | 10% |

117. Explicar por qué la programación en el émbito del mantenimiento puede ser más difícil de llevar a cabo que el desarrollo de un nuevo sistema. ¿Cúales podrían ser las características deseables de un buen programador de mantenimiento?
118. Examinar un programa grande correspondiente a alguno de los proyectos de las asignaturas en que estás matriculado ¿Qué se debe agregar a la documentación para que alguien más pueda mantenerlo? Discutir los *pros* y *cons* de escribir esta documentación suplementaria cuando se desarrolla el programa.
119. Solicitar a un amigo una copia de un programa grande (más de 1000 líneas de código). En lo posible conviene seleccionar un programa con el que no se está familiarizado. ¿Cómo de útil resulta ser la documentación del programa? Comparar el código con la documentación; ¿Cómo de exacta es la documentación? Si estuviésemos asignados para mantener este programa, qué documentación adicional nos gustaría ver? ¿En qué medida el tamaño del programa afecta nuestra habilidad para mantenerlo?

120. Como en el ejercicio anterior, examinar el código de un programa que no hayamos realizado. Suponer que se quiera hacer un cambio de código y que se debe realizar una prueba de regresión sobre el resultado para asegurar que el programa todavía se ejecuta correctamente. ¿Existen y están disponibles para su uso los datos de prueba y un guión de prueba? Discutir la necesidad de retener juegos de datos y guiones de prueba formales para propósitos de mantenimiento.
121. Explicar por qué los componentes de una sola entrada y una sola salida hacen la comprobación más fácil durante el mantenimiento.
122. Repasar lasa características del buen diseño de software. Para cada una, explicar si ayudará o perjudicará en caso de un posterior rejuvenecimiento del software.
123. ¿El software del proyecto Ariane-5 se clasifica como sistema -S, -P o -E?
124. ¿Los números ciclomáticos (McCabe) permiten formar una clasificación de componentes según calidad? Es decir, ¿Siempre podemos determinar si un componente es más complejo que otro? Nombrar algunos aspectos de la complejidad del software que no se determinan mediante el número ciclomático.
125. Para el siguiente código, determinar su número ciclomático utilizando tres métodos diferentes:

```
1  public void drawScore(int n){
2    while(numdigitos>0){
3      score[numdigitos].erase();
4    }
5    if(n==0){
6        //liberar memoria del objeto
7      delete(score[numdigitos]);
8      score[numdigitos]= new Displayable(digitos[0]);
9      score[numdigitos].move(Point((700-numdigitos*18),40));
10     score[numdigitos].draw();
11     numdigitos++;
12   }
13   while(n){
14     int resto= n % 10;
15     delete(score[numdigitos]);
16     score[numdigitos]= new Displayable(digitos[resto]);
17     score[numdigitos].move(Point((700-numdigitos*18),40));
18     score[numdigitos].draw();
19     n= n/10;
20     numdigitos++;
21   }
22 }
```

126. La siguiente es una lista de criterio funcional de control de versión y configuración para las herramientas de gestión de configuración de una agencia británica. Explicar cómo contribuye cada factor a la facilidad de mantenimiento:
    (a) Registrar versiones y las referencias a ellas
    (b) Recuperar cualquier versión a demanda
    (c) Registrar las relaciones
    (d) Registrar las relaciones entre versiones a las que la herramienta controla el acceso y aquellas para las que no lo hace
    (e) Controlar la seguridad y las autorizaciones para grabar o registrar
    (f) Registrar los cambios que se hagan a un archivo
    (g) Registrar el estado de una versión
    (h) Relacionar a una herramienta de control de proyecto
    (i) Producir informes
    (j) Controlar las ediciones
    (k) Controlarse
    (l) Almacenar y recuperar los archivos que no se utilizan frecuentemente

127. Suponer que se está realizando el mantenimiento de un sistema software de seguridad crítica, bastante grande. Se utiliza un modelo, como el de Porter y Selby (ver más abajo) para determinar cuáles son los componentes que es más probable que fallen. Entonces, se examinan esos componentes cuidadosamente identificados y se realiza el mantenimiento preventivo y perfectivo de cada uno de ellos. Poco después el sistema experimenta un falo catastrófico con severas consecuencias para la vida de personas y la propiedad. El motivo del fallo resulta ser un componente que no fue identificado por el modelo ¿Se le pueden exigir responsabilidades legales al equipo de mantenimiento del sistema por haber dejado de analizar los otros componentes?

Notas:

- Porter y Selby (1990) usaron una técnica estadística denominada *anaálisis de árbol de clasificación* para identificar aquellas medidas del producto que actúan como mejores predictores de los errores de la interfaz que probablemente aparezcan durante el mantenimiento:
  1. Entre 4 y 8 revisiones durante el diseño y por lo menos 15 enlaces de datos sugieren que es probable la existencia de defectos en el interfaz.
  2. Los problemas de la interfaz son probables en un componente cuya función primaria es administrar archivos donde ha habido por lo menos 9 revisiones durante el diseño.
  3. Estas sugerencias son particulares para cada juego de datos y no están pensadas como pautas generales para cualquier organización
  4. La técnica puede aplicarse a cualquier base de datos que contenga la información sobre esta clase de medidas.
- El informe de investigación después de la explosión del Ariane-5 señala que los desarrolladores se concentraron en mitigar los fallos aleatorios. Es decir, la guía proporcionada a los desarrolladores del sistema de referencia inercial, como de todo el software, era detener el procesador si se detectaba cualquier excepción. Cuando el sistema de referencia inercial falló, lo hizo debido a un defecto de diseño, no como consecuencia de un fallo al azar, por tanto, el cese del proceso era correcto pero impropio según la misión del cohete. El siguiente paso para el software de Ariane conlleva el cambio de la estrategia de tratamiento de fallos y la implementación de una serie de perfeccionamientos preventivos. Está claro que la ESA no puede enviar otro cohete cada vez que deba probarse un cambio realizado en el software, por lo tanto, la prueba de los cambios producidos en el mantenimiento implica una compleja serie de simulaciones para evaluar los efectos del código nuevo o cambiado. De manera similar, debido a que todo software, como el del sistema de referencia inercial tiene varias versiones, entonces debe aplicarse el control de cambios y gestión de la configuración para asegurar que los cambios exitosos a una versión no degraden inadvertidamente la funcionalidad o el desempeño de otra.