**Inicialización de variables**

* ~~¿Comprueba cada rutina la validez de los parámetros de entrada?~~
* ~~¿Declara el código las variables cerca de donde se utilizan por primera vez?~~
* ~~¿El código inicializa las variables a medida que se declaran, si es posible?~~
* ~~¿Inicializa el código las variables cerca de donde se utilizan por primera vez, si no es posible declararlas e inicializarlas al mismo tiempo?~~
* ~~¿Se inicializan correctamente los contadores y acumuladores y, si es necesario, se reinicializan cada vez que se utilizan?~~
* ~~¿Se reinicializan correctamente las variables en el código que se ejecuta repetidamente?~~
* ¿Compila el código sin advertencias del compilador? (¿Y ha activado todas las advertencias disponibles?)
* Si su lenguaje utiliza declaraciones implícitas, ¿ha compensado los problemas que causan? Otras cuestiones generales sobre el uso de datos
* ¿Tienen todas las variables el ámbito más pequeño posible?
* ~~¿Las referencias a variables están lo más juntas posible, tanto desde cada referencia a una variable a la siguiente referencia como en tiempo total de vida?~~
* ¿Se corresponden las estructuras de control con los tipos de datos?
* ¿Se utilizan todas las variables declaradas?
* ~~¿Están todas las variables enlazadas en los momentos adecuados, es decir, se está logrando un equilibrio consciente entre la flexibilidad del enlace tardío y la mayor complejidad asociada al enlace tardío?~~
* ~~¿Tiene cada variable un único propósito?~~
* ~~¿Es explícito el significado de cada variable, sin significados ocultos?~~

**Tipos de datos abstractos**

* ~~¿Has pensado en las clases de tu programa como tipos de datos abstractos y has evaluado sus interfaces desde ese punto de vista?~~

**Abstracción**

* ~~¿Tiene la clase un propósito central?~~
* ~~¿Está bien nombrada la clase y su nombre describe su propósito central?~~
* ~~¿Presenta la interfaz de la clase una abstracción coherente?~~
* ~~¿La interfaz de la clase hace obvio cómo se debe utilizar la clase?~~
* ~~¿Es la interfaz de la clase lo suficientemente abstracta como para no tener que pensar en cómo se implementan sus servicios?~~
* ¿Puedes tratar la clase como una caja negra?
* ¿Son los servicios de la clase lo suficientemente completos como para que otras clases no tengan que inmiscuirse en sus datos internos?
* ¿Se ha movido información no relacionada fuera de la clase?
* ¿Has pensado en subdividir la clase en clases componentes, y la has subdividido todo lo que has podido?
* ¿Está preservando la integridad de la interfaz de la clase a medida que la modifica?

**Encapsulación**

* ~~¿La clase minimiza la accesibilidad a sus miembros?~~
* ~~¿Evita la clase exponer los datos de sus miembros?~~
* ~~¿Oculta la clase sus detalles de implementación de otras clases tanto como permite el lenguaje de programación?~~
* ¿Evita la clase hacer suposiciones sobre sus usuarios, incluidas sus clases derivadas?
* ¿Es la clase independiente de otras clases? ¿Está débilmente acoplada?

**Herencia**

* ~~¿Se utiliza la herencia sólo para modelar relaciones "es a", es decir, las clases derivadas se adhieren al Principio de Sustitución de Liskov?~~
* ¿Describe la documentación de la clase la estrategia de herencia?
* ¿Evitan las clases derivadas "sobrescribir" rutinas no sobrescribibles?
* ~~¿Están las interfaces, datos y comportamientos comunes lo más arriba posible en el árbol de herencia?~~
* ~~¿Son los árboles de herencia bastante poco profundos?~~
* ~~¿Son todos los miembros de datos de la clase base privados en lugar de protegidos?~~

**Otras cuestiones de implementación**

* ¿Contiene la clase unos siete miembros de datos o menos?
* ¿La clase minimiza las llamadas de rutina directas e indirectas a otras clases?
* ¿Colabora la clase con otras clases sólo en la medida absolutamente necesaria?
* ~~¿Se inicializan todos los datos de los miembros en el constructor?~~
* ¿Está la clase diseñada para ser utilizada como copias profundas en lugar de superficiales, a menos que exista una razón medida para crear copias superficiales? Cuestiones específicas del lenguaje
* ¿Ha investigado las cuestiones específicas del lenguaje para las clases en su lenguaje de programación específico?

**Sentencias if-then**

* ~~¿Está clara la ruta nominal a través del código?~~
* ~~¿Se bifurcan correctamente las pruebas if-then en caso de igualdad?~~
* ¿Está presente y documentada la cláusula else?
* ¿Es correcta la cláusula else?
* ~~¿Se utilizan correctamente -no al revés- las cláusulas if y else?~~
* ~~¿Sigue el caso normal la cláusula if en lugar de la cláusula else?~~

**Cadenas if-then-else-if**

* ¿Se encapsulan las pruebas complicadas en llamadas a funciones booleanas?
* ¿Se prueban primero los casos más comunes?
* ~~¿Se cubren todos los casos?~~
* ~~¿Es la cadena if-then-else-if la mejor implementación-mejor que una sentencia case?~~

**Sentencias case**

* ~~¿Están los casos ordenados de forma significativa?~~
* ¿Son sencillas las acciones para cada caso-llamando a otras rutinas si es necesario?
* ¿Prueba la sentencia case una variable real, no una falsa inventada únicamente para usar y abusar de la sentencia case?
* ~~¿Es legítimo el uso de la cláusula default?~~
* ¿Se utiliza la cláusula default para detectar e informar de casos inesperados? En C, C++ o Java, ¿el final de cada case tiene un break?

**Bucles Selección y creación de bucles**

* ~~¿Se utiliza un bucle while en lugar de un bucle for, si procede?~~
* ¿Se ha creado el bucle desde dentro hacia fuera? Introducción del bucle
* ¿Se introduce el bucle desde arriba?
* ¿Está el código de inicialización directamente antes del bucle?
* Si el bucle es un bucle infinito o un bucle de eventos, ¿está construido limpiamente en lugar de utilizar una chapuza como for i = 1 a 9999?
* Si el bucle es un bucle for de C++, C o Java, ¿se reserva la cabecera del bucle para el código de control del bucle?
* Dentro del bucle ¿Utiliza el bucle { y } o sus equivalentes para encerrar el cuerpo del bucle y evitar problemas derivados de modificaciones inadecuadas?
* ~~¿Tiene algo el cuerpo del bucle? No está vacío?~~
* ~~¿Están agrupadas las tareas domésticas, al principio o al final del bucle?~~
* ¿Realiza el bucle una y sólo una función, como hace una rutina bien definida?
* ~~¿Es el bucle lo suficientemente corto como para verlo todo a la vez?~~
* ¿Está el bucle anidado a tres niveles o menos?
* ¿Se ha trasladado el contenido del bucle largo a su propia rutina?
* Si el bucle es largo, ¿es especialmente claro? Puntos clave 389 Índices de bucle
* Si el bucle es un bucle for, ¿evita el código de su interior jugar con el índice del bucle?
* ¿Se utiliza una variable para guardar valores importantes del índice del bucle en lugar de utilizar el índice del bucle fuera del bucle?
* ~~¿Es el índice del bucle un tipo ordinal o un tipo enumerado-no de coma flotante?~~
* ¿Tiene el índice de bucle un nombre significativo?
* ~~¿Evita el bucle el cruce de índices? Salida del bucle~~
* ~~¿Termina el bucle en todas las condiciones posibles?~~
* ¿Utiliza el bucle contadores de seguridad-si ha instituido una norma de contadores de seguridad? ¿Es obvia la condición de finalización del bucle?
* ~~Si se utilizan break o continue, ¿son correctos?~~

**Rutinas de alta calidad Cuestiones generales**

* ~~¿Es suficiente el motivo para crear la rutina?~~
* ~~¿Se han incluido en rutinas propias todas las partes de la rutina que se beneficiarían de su inclusión en rutinas propias?~~
* ~~¿Es el nombre de la rutina un nombre fuerte y claro del verbo más objeto para un procedimiento o una descripción del valor de retorno para una función?~~
* ~~¿El nombre de la rutina describe todo lo que hace la rutina?~~
* ¿Ha establecido convenciones de nomenclatura para las operaciones habituales?
* ~~¿Tiene la rutina una cohesión funcional fuerte -hace una y sólo una cosa y la hace bien?~~
* ~~¿Las rutinas tienen un acoplamiento flexible: las conexiones de la rutina con otras rutinas son pequeñas, íntimas, visibles y flexibles?~~
* ~~¿Está la longitud de la rutina determinada de forma natural por su función y lógica, en lugar de por un estándar de codificación artificial?~~

**Cuestiones relativas al paso de parámetros**

* ¿Presenta la lista de parámetros de la rutina, en su conjunto, una abstracción de interfaz coherente?
* ~~¿Están los parámetros de la rutina en un orden razonable, incluyendo la coincidencia con el orden de los parámetros en rutinas similares?~~
* ~~¿Están documentadas las suposiciones de la interfaz?~~
* ~~¿Tiene la rutina siete o menos parámetros?~~
* ~~¿Se utiliza cada parámetro de entrada?~~
* ~~¿Se utiliza cada parámetro de salida?~~
* ¿Evita la rutina utilizar los parámetros de entrada como variables de trabajo?
* ~~Si la rutina es una función, ¿devuelve un valor válido en todas las circunstancias posibles?~~

**Consideraciones generales sobre nombres**

* ~~¿Describe el nombre de forma completa y precisa lo que representa la variable?~~
* ~~¿Se refiere el nombre al problema del mundo real y no a la solución del lenguaje de programación?~~
* ~~¿Es el nombre lo suficientemente largo como para no tener que descifrarlo?~~
* ~~¿Están los calificadores de valor calculado, si los hay, al final del nombre?~~
* ~~¿Utiliza el nombre Count o Index en lugar de Num?~~

**Denominación de tipos específicos de datos**

* ¿Tienen sentido los nombres de índices de bucles (algo distinto de i, j o k si el bucle tiene más de una o dos líneas de longitud o está anidado)?
* ~~¿Se han renombrado todas las variables "temporales" a algo más significativo?~~
* ~~¿Se han nombrado las variables booleanas de forma que quede claro su significado cuando son verdaderas?~~
* ¿Incluyen los nombres de tipo enumerado un prefijo o sufijo que indique la categoría -por ejemplo, Color\_ para Color\_Rojo, Color\_Verde, Color\_Azul, etc.?
* ¿Se nombran las constantes por las entidades abstractas que representan y no por los números a los que se refieren?

**Convenciones de nomenclatura**

* ~~¿Distingue la convención entre datos locales, de clase y globales?~~
* ¿Distingue la convención entre nombres de tipos, constantes con nombre, tipos enumerados y variables?
* ¿Identifica la convención los parámetros sólo de entrada a rutinas en lenguajes que no los imponen?
* ¿Es la convención lo más compatible posible con las convenciones estándar del lenguaje?
* ~~¿Están los nombres formateados para facilitar su lectura?~~

**Nombres cortos**

* ~~¿Utiliza el código nombres largos (a menos que sea necesario utilizar nombres cortos)?~~
* ~~¿Evita el código las abreviaturas que ahorran un solo carácter?~~
* ~~¿Se abrevian todas las palabras de forma coherente?~~
* ~~¿Se pueden pronunciar los nombres?~~
* ~~¿Se evitan los nombres que puedan ser mal leídos o pronunciados?~~
* ¿Se documentan los nombres abreviados en las tablas de traducción?

**Problemas comunes de denominación:**

¿Se han evitado

* ~~...nombres que induzcan a error?~~
* ~~...nombres con significados similares?~~
* ~~...nombres que sólo se diferencian por uno o dos caracteres?~~
* ~~...nombres que suenan parecido?~~
* ~~...nombres que utilizan números?~~
* ~~...nombres mal escritos intencionadamente para acortarlos?~~
* ~~...nombres que suelen escribirse mal en inglés?~~
* ~~...nombres que entran en conflicto con nombres de rutinas de biblioteca estándar o con nombres de variables predefinidas?~~
* ~~...nombres totalmente arbitrarios?~~
* ~~...caracteres difíciles de leer?~~

**Números de datos fundamentales en general**

* ~~Evita números mágicos~~
* ~~Anticipa divisiones entre 0~~
* ~~Las conversiones de tipos son obvias~~
* Si se utilizan variables con dos tipos diferentes en la misma expresión, ¿se evaluará la expresión como se pretende?
* Si variables con dos tipos diferentes son usadas en la misma expresión,
* ~~¿Evita el código las comparaciones de tipo mixto?~~
* ¿Compila el programa sin advertencias?

**Enteros**

* ¿Las expresiones que utilizan la división de enteros funcionan como deben?
* ¿Evitan las expresiones con números enteros los problemas de desbordamiento de enteros?

**Números flotantes**

* ¿Evita el código sumas y restas en números con magnitudes muy diferentes?
* ¿Evita el código sistemáticamente errores de redondeo?
* ¿Evita el código comparar números en coma flotante para comprobar su igualdad?

**Caracteres y cadenas**

* ~~¿Evita el código los caracteres mágicos y las cadenas?~~
* ¿Trata el código C de forma diferente los punteros a cadenas y las matrices de caracteres?
* ¿Sigue el código C la convención de declarar cadenas de longitud CON STANT+1?
* ¿Utiliza el código C matrices de caracteres en lugar de punteros, cuando es apropiado?
* ¿Inicializa el código C las cadenas a NULLs para evitar cadenas infinitas?
* ¿Utiliza el código C strncpy() en lugar de strcpy()? Y strncat() y strncmp()?

**Variables booleanas**

* ~~¿Utiliza el programa variables booleanas adicionales para documentar pruebas condicionales?~~
* ~~¿Utiliza el programa variables booleanas adicionales para simplificar las pruebas condicionales?~~

**Tipos enumerados**

* ¿Utiliza el programa tipos enumerados en lugar de constantes con nombre por su mejor legibilidad, fiabilidad y modificabilidad?
* ¿Utiliza el programa tipos enumerados en lugar de variables booleanas cuando el uso de una variable no se puede capturar completamente con verdadero y falso?
* ¿Las pruebas que utilizan tipos enumerados comprueban los valores no válidos?
* ¿Se reserva la primera entrada de un tipo enumerado para "no válido"?

**Constantes con nombre**

* ~~¿Utiliza el programa constantes con nombre para las declaraciones de datos y los límites de bucle en lugar de números mágicos?~~
* ~~¿Se han utilizado las constantes con nombre de forma coherente-no se utilizan como constantes con nombre en algunos lugares y como literales en otros?~~

**Arrays**

* ~~¿Están todos los índices de los arrays dentro de los límites del mismo?~~
* ~~¿Las referencias de los arrays no contienen errores de separación?~~
* ¿Están todos los subíndices de los arrays multidimensionales en el orden correcto?
* En los bucles anidados, ¿se utiliza la variable correcta como subíndice del array, evitando el cruce bucle-índice?

**Creación de tipos**

* ¿Utiliza el programa un tipo diferente para cada tipo de dato que pueda cambiar?
* ¿Están los nombres de los tipos orientados hacia las entidades del mundo real que representan los tipos en lugar de hacia los tipos del lenguaje de programación?
* ¿Son los nombres de los tipos lo suficientemente descriptivos como para ayudar a documentar las declaraciones de datos?
* ¿Ha evitado redefinir tipos predefinidos?
* ¿Ha considerado crear una nueva clase en lugar de simplemente redefinir un tipo?

**Organización del código de línea recta**

* ¿El código hace obvias las dependencias entre sentencias?
* ~~¿Los nombres de las rutinas hacen obvias las dependencias?~~
* ¿Los parámetros de las rutinas hacen obvias las dependencias?
* ~~¿Los comentarios describen las dependencias que de otro modo no quedarían claras?~~
* ¿Se han utilizado variables de mantenimiento para comprobar las dependencias secuenciales en secciones críticas del código?
* ~~¿Se lee el código de arriba abajo?~~
* ~~¿Están agrupadas las sentencias relacionadas?~~
* ~~¿Se han trasladado grupos de sentencias relativamente independientes a sus propias rutinas?~~

**Buena técnica de comentarios**

**Generalidades**

* ~~¿Puede alguien agarrar el código y empezar a entenderlo inmediatamente?~~
* ~~¿Los comentarios explican la intención del código o resumen lo que hace el código, en lugar de limitarse a repetir el código?~~
* ¿Se utiliza el Proceso de Programación con Pseudocódigo para reducir el tiempo de comentario?
* ~~¿Se ha reescrito código complicado en lugar de comentarlo?~~
* ~~¿Están actualizados los comentarios?~~
* ~~¿Son claros y correctos los comentarios?~~
* ~~¿El estilo de los comentarios permite modificarlos fácilmente?~~

**Enunciados y párrafos**

* ¿Evita el código los comentarios al final de línea?
* ~~¿Se centran los comentarios en el por qué y no en el cómo?~~
* ~~¿Los comentarios preparan al lector para el código que sigue?~~
* ~~¿Cada comentario cuenta? ¿Se han eliminado o mejorado los comentarios redundantes, superfluos y autoindulgentes?~~
* ¿Están documentadas las sorpresas?
* ~~¿Se han evitado las abreviaturas?~~
* ¿Está clara la distinción entre comentarios mayores y menores?
* ~~¿Se comenta el código que soluciona un error o una característica no documentada?~~

**Declaraciones de datos**

* ¿Se comentan las unidades en las declaraciones de datos?
* ¿Se comentan los rangos de valores en los datos numéricos?
* ¿Se comentan los significados codificados?
* ¿Se comentan las limitaciones de los datos de entrada?
* ¿Están documentados los indicadores a nivel de bits?
* ¿Se ha comentado cada variable global donde se declara?
* ¿Se ha identificado cada variable global como tal en cada uso, mediante una convención de nomenclatura, un comentario o ambos?
* ~~¿Se han sustituido los números mágicos por constantes o variables con nombre en lugar de simplemente documentarlas?~~

**Estructuras de control**

* ¿Se comenta cada sentencia de control?
* ¿Se comentan los extremos de las estructuras de control largas o complejas o, cuando es posible, se simplifican para que no necesiten comentarios?

**Rutinas**

* ~~¿Se comenta la finalidad de cada rutina?~~
* ~~Si procede, ¿se comentan otros datos sobre cada rutina, como los datos de entrada y salida, los supuestos de la interfaz, las limitaciones, las correcciones de errores, los efectos globales y las fuentes de los algoritmos?~~

**Ficheros, clases y programas**

* ¿Dispone el programa de un documento breve, como el descrito en el Paradigma Libro, que ofrezca una visión general de cómo está organizado el programa?
* ¿Se describe la finalidad de cada archivo?
* ¿Están el nombre, la dirección de correo electrónico y el número de teléfono del autor en el listado?