**Clases**

**Tipos de datos abstractos**

¿Ha pensado en las clases de su programa como Tipos de datos abstractos y ha evaluado sus interfaces desde ese punto de vista? √

**Abstracción**

¿La clase tiene un propósito central? √

¿La clase está bien nombrada y su nombre describe su propósito central? √

¿La interfaz de la clase presenta una abstracción consistente? √

¿La interfaz de la clase hace obvio cómo debe usar la clase? √

¿Es la interfaz de la clase lo suficientemente abstracta como para no tener que pensar? √

¿Cómo se implementan sus servicios? ¿Puedes tratar la clase como una caja negra? √

¿Los servicios de la clase son lo suficientemente completos como para que otras clases no tengan que entrometerse con sus datos internos?

¿Se ha sacado de la clase información no relacionada? √

¿Has pensado en subdividir la clase en clases componentes y la has subdividido tanto como puedas? √

¿Preserva la integridad de la interfaz de la clase a medida que modifica la clase? √

**Encapsulamiento**

¿La clase minimiza la accesibilidad a sus miembros? √

¿La clase evita exponer datos de miembros?

¿Oculta la clase sus detalles de implementación de otras clases tanto como lo permite el lenguaje de programación?

¿La clase evita hacer suposiciones sobre sus usuarios, incluidas sus clases derivadas? √

¿Es la clase independiente de otras clases? ¿Está suelto?

**Herencia**

¿La herencia se usa solo para modelar relaciones "es a"?

¿La documentación de la clase describe la estrategia de herencia?

¿Las clases derivadas se adhieren al Principio de sustitución de Liskov?

¿Las clases derivadas evitan "anular" rutinas no reemplazables?

¿Las interfaces, los datos y el comportamiento comunes son lo más altos posible en el árbol de herencia?

¿Son los árboles de herencia bastante poco profundos?

¿Todos los miembros de datos en la clase base son privados en lugar de protegidos?

**Otros problemas de implementación**

¿La clase contiene aproximadamente siete miembros de datos o menos? √

¿La clase minimiza las llamadas de rutina directas e indirectas a otras clases? √

¿La clase colabora con otras clases solo en la medida absolutamente necesaria? √

¿Todos los datos de miembros se inicializan en el constructor?

¿La clase está diseñada para usarse como copias profundas en lugar de copias superficiales a menos que haya una razón medida para crear copias superficiales? √

**Problemas específicos del idioma**

¿Ha investigado los problemas específicos del idioma para las clases en su lenguaje de programación específico? √

**Rutinas**

**Problemas generales**

¿Es suficiente la razón para crear la rutina? √

¿Todas las partes de la rutina que se beneficiarían de ser puestas en rutinas propias se han puesto en rutinas propias? √

¿Es el nombre de la rutina un nombre fuerte y claro de 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 operaciones comunes?

¿Tiene la rutina una cohesión fuerte y funcional: hacer una y solo una cosa y hacerlo bien? √

¿Las rutinas tienen un acoplamiento flojo? ¿Las conexiones de la rutina con otras rutinas son pequeñas, íntimas, visibles y flexibles? √

¿La duración de la rutina está determinada naturalmente por su función y lógica, más que por un estándar de codificación artificial? √

**Problemas de paso de parámetros**

¿La lista de parámetros de la rutina, tomada en su conjunto, presenta una abstracción de interfaz consistente? √

¿Están los parámetros de la rutina en un orden razonable, incluyendo la coincidencia del orden de los parámetros en rutinas similares? √

¿Están documentados los supuestos de la interfaz? √

¿La rutina tiene siete o menos parámetros? √

¿Se utiliza cada parámetro de entrada? √

¿Se utiliza cada parámetro de salida? √

¿La rutina evita usar parámetros de entrada como variables de trabajo? √

Si la rutina es una función, ¿devuelve un valor válido en todas las circunstancias posibles? √

**Uso de datos**

**Inicializando Variables**

¿Cada rutina verifica la validez de los parámetros de entrada? √

¿El código declara variables cercanas al lugar donde se usaron por primera vez? √

¿El código inicializa las variables como se declaran, si es posible? √

¿El código inicializa las variables cerca de donde se usaron por primera vez, si no es posible declararlas e inicializarlas al mismo tiempo? √

¿Los contadores y acumuladores se inicializan correctamente y, si es necesario, se reinician cada vez que se usan? √

¿Las variables se reinicializan correctamente en el código que se ejecuta repetidamente? √

¿El código se compila sin advertencias del compilador? √

Si su lenguaje usa declaraciones implícitas, ¿ha compensado los problemas que causan?

**Otros problemas generales en el uso de datos**

¿Todas las variables tienen el alcance más pequeño posible?

¿Las referencias a variables están tan juntas como sea posible, tanto de cada referencia a una variable a la siguiente como en tiempo total de vida? √

¿Las estructuras de control corresponden a los tipos de datos? √

¿Se están utilizando todas las variables declaradas? √

¿Están todas las variables vinculadas en los momentos apropiados, es decir, logrando un equilibrio consciente entre la flexibilidad de la unión tardía y la mayor complejidad asociada con la unión tardía?

¿Cada variable tiene un solo propósito? √

¿El significado de cada variable es explícito, sin significados ocultos? √

**Nombramiento de variables**

**Consideraciones generales de nomenclatura**

¿El nombre describe completa y exactamente lo que representa la variable? √

¿El nombre se refiere al problema del mundo real más que a la solución del lenguaje de programación? √

¿El nombre es lo suficientemente largo como para no tener que descifrarlo? √

¿Los calificadores de valor calculado, si los hay, al final del nombre?

¿El nombre usa Count o Index en lugar de Num? √

**Nombrar tipos específicos de datos**

¿Son significativos los nombres de índice de bucle (algo diferente de i, j o ​​k si el bucle tiene más de una o dos líneas de largo o está anidado)? √

¿Se ha cambiado el nombre de todas las variables "temporales" por algo más significativo? √

¿Se nombran las variables booleanas de modo que sus significados cuando son verdaderos sean claros? √

¿Los nombres de tipo enumerado incluyen un prefijo o sufijo que indica la categoría, por ejemplo, Color para Color Red, Color\_Green, Color\_Blue, etc.?

¿Se nombran las constantes nombradas para las entidades abstractas que representan en lugar de los números a los que se refieren?

**Convenciones de nombres**

¿La convención distingue entre datos locales, de clase y globales?

¿La convención distingue entre nombres de tipos, constantes con nombre, tipos enumerados y variables?

¿La convención identifica parámetros de solo entrada para rutinas en idiomas que no los imponen?

¿La convención es lo más compatible posible con las convenciones estándar para el lenguaje?

¿Los nombres están formateados para facilitar la lectura?

**Nombres cortos**

¿El código usa nombres largos (a menos que sea necesario usar nombres cortos)? √

¿El código evita las abreviaturas que guardan solo un carácter? √

¿Se abrevian todas las palabras de manera consistente? √

¿Son pronunciables los nombres? √

¿Se evitan los nombres que se pueden pronunciar mal? √

¿Se documentan los nombres cortos en las tablas de traducción?

**Problemas comunes de nombres: ¿ha evitado ...**

... ¿nombres que son engañosos? √

... ¿nombres con significados similares?

... ¿nombres que son diferentes solo por uno o dos caracteres?

... nombres que suenan similares?

... ¿nombres que usan números? √

... ¿nombres mal escritos intencionalmente para acortarlos? √

... ¿nombres que generalmente están mal escritos en inglés? √

... ¿nombres que entran en conflicto con los nombres de rutina de biblioteca estándar o con nombres de variables predefinidos? √

... nombres totalmente arbitrarios? √

... ¿personajes difíciles de leer? √

**Tipos de datos fundamentales**

**Números en general**

¿El código evita los números mágicos? √

¿El código anticipa errores de división por cero? √

¿Son obvias las conversiones de tipos? √

Si se usan variables con dos tipos diferentes en la misma expresión, ¿se evaluará la expresión tal como lo desea? √

¿El código evita las comparaciones de tipo mixto? √

¿El programa se compila sin advertencias? √

**Enteros**

¿Las expresiones que usan la división de enteros funcionan de la manera en que están destinadas? √

¿Las expresiones enteras evitan problemas de desbordamiento de enteros?

**Números de punto flotante**

¿El código evita sumas y restas en números con magnitudes muy diferentes? √

¿El código previene sistemáticamente errores de redondeo?

¿El código evita comparar números de punto flotante para igualdad? √

**Caracteres y cadenas**

¿El código evita caracteres mágicos y cadenas? √

¿Las referencias a cadenas están libres de errores off-by-one?

¿El código C trata los punteros de cadena y las matrices de caracteres de manera diferente?

¿Sigue el código C la convención de declarar cadenas como longitud constante + 1?

¿El código C utiliza matrices de caracteres en lugar de punteros, cuando corresponde?

¿El código C inicializa cadenas a NULL para evitar cadenas interminables?

¿El código C usa 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 para mejorar su legibilidad, fiabilidad y modificabilidad?

¿Utiliza el programa tipos enumerados en lugar de variables booleanas cuando el uso de una variable no puede capturarse completamente con VERDADERO y FALSO?

¿Las pruebas que utilizan tipos enumerados prueban valores no válidos?

¿La primera entrada en un tipo enumerado está reservada para "inválido"?

**Constantes nombradas**

¿Utiliza el programa constante con nombre para declaraciones de datos y límites de bucle en lugar de números mágicos?

¿Se han usado consistentemente las constantes con nombre, no constantes con nombre en algunos lugares, literales en otros?

**Matrices**

¿Están todos los índices de la matriz dentro de los límites de la matriz? √

¿Están las referencias de matriz libres de errores fuera de uno? √

¿Están todos los subíndices en matrices multidimensionales en el orden correcto? √

En bucles anidados, ¿se usa la variable correcta como subíndice de matriz, evitando la conversación cruzada de índice de bucle? √

**Crear tipos**

¿Utiliza el programa un tipo diferente para cada tipo de datos que pueden cambiar?

¿Están los nombres de tipo orientados hacia las entidades del mundo real que representan los tipos en lugar de hacia tipos de lenguaje de programación?

¿Son los nombres de tipo lo suficientemente descriptivos como para ayudar a documentar las declaraciones de datos?

¿Has evitado redefinir tipos predefinidos?

¿Ha considerado crear una nueva clase en lugar de simplemente redefinir un tipo?

**Organizing Straight-Line Code**

¿El código hace obvias las dependencias entre las declaraciones? √

¿Los nombres de las rutinas hacen obvias las dependencias? √

¿Los parámetros de las rutinas hacen obvias las dependencias? √

¿Los comentarios describen dependencias que de otro modo no estarían claras? √

¿Se han utilizado variables de limpieza para verificar dependencias secuenciales en secciones críticas de código?

¿El código se lee de arriba a abajo? √

¿Se agrupan las declaraciones relacionadas? √

¿Se han movido grupos relativamente independientes de declaraciones a sus propias rutinas? √

**Condicionales**

**declaraciones if-then**

¿Está clara la ruta nominal a través del código? √

¿Las pruebas if-then se ramifican correctamente en igualdad? √

¿La cláusula else está presente y documentada? √

¿Es correcta la cláusula else? √

¿Las cláusulas if y else se usan correctamente, no se invierten? √

¿Sigue el caso normal el if en lugar del otro? √

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

¿Se encapsulan pruebas complicadas en llamadas a funciones booleanas?

¿Se prueban primero los casos más comunes? √

¿Están cubiertos todos los casos? √

¿Es la cadena if-then-else-if la mejor implementación, mejor que una declaración de caso? √

**Declaraciones de case**

¿Se ordenan los casos de manera significativa? √

¿Son simples las acciones para cada caso, llamando a otras rutinas si es necesario? √

¿La declaración del caso prueba una variable real, no una falsa que está hecha únicamente para usar y abusar de la declaración del caso? √

¿Es legítimo el uso de la cláusula predeterminada? √

¿Se utiliza la cláusula predeterminada para detectar e informar casos inesperados? √

En C, C ++ o Java, ¿el final de cada caso tiene un descanso? √

**Ciclos**

**Selección de bucle y creación**

¿Se usa un ciclo while en lugar de un ciclo for, si corresponde? √

¿El bucle se creó de adentro hacia afuera? √

**Entrar en el bucle**

¿Se ingresa 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, ¿se construye limpiamente en lugar de usar un kludge como por ejemplo para i = 1 a 9999?

Si el bucle es un C ++, C o Java para el bucle, ¿está el encabezado del bucle reservado para el código de control del bucle? √

**Dentro del bucle**

¿El bucle usa {y} o su equivalente para evitar problemas derivados de modificaciones incorrectas?

¿El cuerpo del bucle tiene algo dentro? ¿No es vacío? √

¿Están agrupados los que haceres domésticos, al principio o al final del ciclo? √

¿El bucle realiza una y solo una función, como lo hace una rutina bien definida? √

¿El bucle es lo suficientemente corto como para verlo todo a la vez? √

¿El bucle está anidado en tres niveles o menos? √

¿Se han trasladado los contenidos de bucles largos a su propia rutina?

Si el ciclo es largo, ¿está especialmente claro? √

**Índices de bucle**

Si el bucle es un bucle for, ¿el código dentro de él evita la simulación con el índice del bucle? √

¿Se utiliza una variable para guardar valores importantes de índice de bucle en lugar de usar el índice de bucle fuera del bucle? √

¿Es el índice del bucle un tipo ordinal o un tipo enumerado, no un punto flotante? √

¿El índice de bucle tiene un nombre significativo? √

¿El ciclo evita la conversación cruzada de índice? √

**Saliendo del bucle**

¿El ciclo termina en todas las condiciones posibles? √

¿Utiliza el bucle contadores de seguridad, si ha instituido un estándar de contador de seguridad? √

¿Es obvia la condición de terminación del bucle? √

Si se usan break o continue, ¿son correctos? √

**Estructuras de control**

¿Las expresiones usan Verdadero y Falso en lugar de 1 y 0? √

¿Los valores booleanos se comparan con Verdadero y Falso implícitamente? √

¿Se comparan los valores numéricos con sus valores de prueba explícitamente? √

¿Se han simplificado las expresiones mediante la adición de nuevas variables booleanas y el uso de funciones booleanas y tablas de decisión?

¿Se expresan positivamente las expresiones booleanas? √

¿Se equilibran los pares de llaves? √

¿Se usan llaves en todas partes donde se necesitan para mayor claridad? √

¿Las expresiones lógicas están completamente entre paréntesis? √

¿Se han escrito las pruebas en orden de línea de números?

¿Las pruebas de Java utilizan el estilo a.equals (b) en lugar de a == b cuando corresponde? √

¿Son obvias las declaraciones nulas? √

¿Se han simplificado las declaraciones anidadas volviendo a probar parte del condicional, convirtiéndolas en declaraciones if-then-else o case, moviendo el código anidado a su propia rutina, convirtiendo a un diseño más orientado a objetos, o mejorando de alguna otra manera? √

Si una rutina tiene un conteo de decisiones de más de 10, ¿hay una buena razón para no rediseñarla? √

**Self-Documenting Code**

**General**

¿Alguien puede recoger el código e inmediatamente comenzar a entenderlo?

¿Los comentarios explican la intención del código o resumen lo que hace el código, en lugar de simplemente repetir el código?

¿Se utiliza el proceso de programación de pseudocódigo para reducir el tiempo de comentario? √

¿Se ha reescrito el código complicado en lugar de comentarlo? √

¿Los comentarios están actualizados? √

¿Los comentarios son claros y correctos? √

¿El estilo de comentario permite modificar los comentarios fácilmente?

**Declaraciones y párrafos**

¿El código evita los comentarios finales?

¿Los comentarios se centran en por qué en lugar de cómo? √

¿Los comentarios preparan al lector para que siga el código?

¿Cuenta cada comentario? Tener redundante, extraño y autocomplaciente

¿Se han eliminado o mejorado comentarios? √

¿Están documentadas las sorpresas?

¿Se han evitado las abreviaturas?

¿Está clara la distinción entre comentarios mayores y menores?

¿Se comenta el código que funciona alrededor de 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 datos numéricos?

¿Se comentan los significados codificados?

¿Se comentan las limitaciones en los datos de entrada?

¿Están las banderas documentadas al nivel de bit?

¿Se ha comentado cada variable global donde se declara?

¿Se ha identificado cada variable global como tal en cada uso, por una convención de nomenclatura, un comentario o ambos?

¿Se reemplazan los números mágicos con constantes o variables con nombre en lugar de simplemente documentarse? √

**Estructuras de Control**

¿Se comenta cada declaración de control? √

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

**Rutinas**

¿Se comenta el propósito de cada rutina? √

¿Se incluyen otros hechos sobre cada rutina en los comentarios, cuando son relevantes, incluyendo datos de entrada y salida, suposiciones de interfaz, limitaciones, correcciones de errores, efectos globales y fuentes de algoritmos? √

**Archivos, clases y programas**

¿Tiene el programa un documento breve como el que se describe en el libro Paradigma que ofrece una visión general de cómo está organizado el programa?

¿Se describe el propósito de cada archivo? √

¿El nombre, la dirección de correo electrónico y el número de teléfono del autor figuran en la lista?