Praktikum Software Engineering 99: Code Inspection Checklist

**Java Code Inspection Checklist**

**1. Variable and Constant Declaration Defects (VC)**

**1. Are descriptive variable and constant names used in accord with naming conventions?**

No, ya que hay variables abreviadas, líneas 18-25, 28, 61, 62, etc...

**2. Are there variables with confusingly similar names?**

Sí, auxb y auxp, líneas 62, 63, 64

**3. Is every variable properly initialized?**

Sí.

**4. Could any non-local variables be made local?**

No hay variables no locales.

**5. Are there literal constants that should be named constants?**

No.

**6. Are there macros that should be constants?**

No hay macros.

**7. Are there variables that should be constants?**

No.

**2. Function Definition Defects (FD)**

**8. Are descriptive function names used in accord with naming conventions?**

Los nombres de las funciones son autodescriptivos y siguen las normas.

**9. Is every function parameter value checked before being used?**

No, ninguna función comprueba sus parámetros. Ejemplo: 41, instertLine

**10. For every function: Does it return the correct value at every function return point?**

Todas la funciones devuelven el valor correcto.

**3. Class Definition Defects (CD)**

**11. Does each class have an appropriate constructor and destructor?**

La clase tiene constructor. La clase no tiene destructor.

**12. For each member of every class: Could access to the member be further restricted?**

No, todos los miembros están bien restringidos.

**13. Do any derived classes have common members that should be in the base class?**

No hay clases derivadas en el código que estamos estudiando.

**14. Can the class inheritance hierarchy be simplified?**

No se puede simplificar más, no hay herencia.

**4. Computation/Numeric Defects (CN)**

**15. Is overflow or underflow possible during a computation?**

No

**16. For each expressions with more than one operator: Are the assumptions about order of evaluation and precedence correct?**

Sí, linea 76 se utiliza solo un tipo de operador así que orden no importa.

**17. Are parentheses used to avoid ambiguity?**

Los paréntesis se utilizan para quitar ambigüedad en línea 76.

**5. Comparison/Relational Defects (CR)**

**18. Are the comparison operators correct?**

Si todos los operadores de comparación son correctos línea 76.

**19. Is each boolean expression correct?**

SI, todas la expresiones booleanas son correctas líneas 76 y 89.

**20. Are there improper and unnoticed side-effects of a comparison?**

No

**6. Control Flow Defects (CF)**

**21. For each loop: Is the best choice of looping constructs used?**

No hay ningún bucle.

**22. Will all loops terminate?**

No hay ningún bucle.

**23. When there are multiple exits from a loop, is each exit necessary and handled properly?**

No hay ningún bucle.

**24. Does each switch statement have a default case?**

No tenemos ningún switch statement.

**25. Are missing switch case break statements correct and marked with a comment?**

No tenemos ningún switch statement.

**26. Is the nesting of loops and branches too deep, and is it correct?**

No hay ningún bucle.

**27. Can any nested if statements be converted into a switch statement?**

No hay ningún nested if statements.

**28. Are null bodied control structures correct and marked with braces or comments?**

En el código no hay ninguna estructura de control con el cuerpo vacío.

**29. Does every function terminate?**

Todas las funciones terminan ya que tienen un return que se alcanzan.

**30. Are goto statements avoided?**

No hay sentencias goto.

**7. Input-Output Defects (IO)**

**31. Have all files been opened before use?**

No hay ficheros de entrada salida.

**32. Are the attributes of the open statement consistent with the use of the file?**

No hay ficheros de entrada salida.

**33. Have all files been closed after use?**

No hay ficheros de entrada salida.

**34. Is buffered data flushed?**

No hay ficheros de entrada salida.

**35. Are there spelling or grammatical errors in any text printed or displayed?**

No hay ficheros de entrada salida.

**36. Are error conditions checked?**

No hay ficheros de entrada salida.

**8. Module Interface Defects (MI)**

**37. Are the number, order, types, and values of parameters in every function call in agreement with the called function’s declaration?**

Si, líneas 35-36, 90-97

**38. Do the values in units agree (e.g., inches versus yards)?**

No se usan unidades

**9. Comment Defects (CM)**

**39. Does every function, class, and file have an appropriate header comment?**

No, la función en linea 54 no tiene ningún comentario

**40. Does every variable or constant declaration have a comment?**

No, líneas 27, 28.

**41. Is the underlying behavior of each function and class expressed in plain language?**

No, hay funciones sin comentario (54).

**42. Is the header comment for each function and class consistent with the behavior of the function or class?**

No, sólo en la clase principal hay comentario, en las funciones no.

**43. Do the comments and code agree?**

Si, líneas 1-16.

**44. Do the comments help in understanding the code?**

Sí, los comentarios escritos ayudan a comprender la clase programada, líneas 1-16.

**45. Are there enough comments in the code**

No.

**46. Are there too many comments in the code?**

No.

**10. Packaging Defects (LP)**

**47. For each file: Does it contain only one class?**

Si

**48. For each function: Is it no more than about 60 lines long?**

Si, ninguna de las funciones tiene más de 60 líneas de longitud.

**49. For each class: Is no more than 2000 lines long (Sun Coding Standard) ?**

No hay ninguna clase con más de 2000 líneas

**11. Modularity Defects (MO)**

**50. Is there a low level of coupling between packages (classes)?**

Si, solo se utilizan getters y setters de otra clase.

**51. Is there a high level of cohesion within each package?**

Si, parser.java tiene un solo uso.

**52. Is there duplicate code that could be replaced by a call to a function that provides the behavior of the duplicate code?**

No, en parser.java no hay codigo duplicado

**53. Are framework classes used where and when appropriate?**

No, no se utilizan Framework classes.

**12. Performance Defects (PE) [Optional]**

**54. Can better data structures or more efficient algorithms be used?**

**55. Are logical tests arranged such that the often successful and inexpensive tests precede the more expensive and less frequently successful tests?**

**56. Can the cost of recomputing a value be reduced by computing it once and storing the results?**

**57. Is every result that is computed and stored actually used?**

**58. Can a computation be moved outside a loop?**

**59. Are there tests within a loop that do not need to be done?**

**60. Can a short loop be unrolled?**

**61. Are there two loops operating on the same data that can be combined into one?**

slightly adapted from the C++ Inpsection Checklist of Christopher Fox, http://www.cs.jmu.edu/users/foxcj/cs555/StdDoc/CppChk.htm.

Copyright 1998 Christopher Fox