# A checklist for design reviews

Manual design reviews are effective in finding smells in design. Use this checklist when you are reviewing UML diagrams (mainly class diagrams) or code. Please note that these are merely indicators of potential problems - you need to dig deeper and analyze further to confirm if they are actual problems or not.

## Abstraction

1. "Encoded strings" or "clumps of primitive types" are used instead of  an abstraction (i.e., a class or an interface)

2. There are one method classes with the following characteristics: a) class name and method names are same b) the class is a stand-alone class and c) there are no other methods or data members

3. Interrelated methods are provided together in the class or interface

4. The class or interface has "too many methods"

5. The members in class or interface is "non-cohesive"

6. The class or interface has only constants

7. The class or interface is empty or has just one or two members with the class "not doing much"

8. The class all of its methods delegating its calls to another class (i.e., the class serves as an "agent class")

9. The class or interface is not used by any other classes in the codebase (i.e., unreachable)

10. The interface or abstract class has no types derived from it

11. Classes or interfaces with the same name are present in different packages/sub-packages

12. Classes or interfaces with the same members are present in the same codebase

## Encapsulation

13. Data member(s) in the class has public access

14. The members (methods or fields) are provided with more lenient access specifiers (e.g., public instead of protected or private access)

15. A class has globally accessible static data member

16. The class returns handle to one or more of its internal data structures through public methods

17. Public method(s) reveal implementation details

18. The class hierarchy exhibits "nested generalizations", i.e., the first level in a inheritance hierarchy factors one generalization and the further levels "multiples out" all possible combinations.

19. The code uses explicit type checks using switch or chained if-else statements

## Modularization

20. The class has only (non-static) data members

21. The members of a class or interface (that ideally should have been localized into that type) are separated and spread across multiple classes or interfaces

22. One or more methods in a class are more interested in another class, potentially showing misplaced methods.

23. The class or interface has huge number of public members

24. The class has method(s) with excessive cyclomatic complexity

24. The class has circular references

25. The class has very high fan-in as well as fan-out

## Hierarchy

26. The code has switch or chained if-else statements for selecting behavior based on encoded type values (e.g., enumerations)

27. The class has a tag field that indicates the flavor of the instance

28. None of the derived classes in a hierarchy override methods from the base class

29. An abstract class or interface has only one concrete type realizing it

30. Sibling classes in a hierarchy share a significant amount of code

31. Derived classes (at the same level) in a hierarchy have duplicate code segments

32. Base class and derived class share duplicate code segments

33. Number of operations added by a derived type is very high (suggesting that some intermediate types between the derived type and its base type could be missing)

34. A base type has large number of immediate derived types

35. The inheritance hierarchy looks like a "list" (i.e., each type has only zero or one derived type)

36. Classes in the inheritance hierarchy do not have any polymorhic methods (i.e., overridable methods)

37. The inheritance hierarchy is very deep

38. Method(s) in a derived class reject a base class method (by providing an empty method body, by throwing an exception, by printing "should not implement" message, or by returning an error value indicating that the method is not supported)

39. The base and derived classes do not share a IS-A relationship

40. A derived type (directly and indirectly) inherits from one or more base types leading to unnecessary inheritance paths in the hierarchy

41. A base type in a hierarchy has a reference to one of its derived types