**CODE & DESIGN SMELLS**

1. **General**
2. Global state
   1. Global variables
   2. Static variables
   3. Static data members
   4. Singletons
   5. Etc.
3. Too many control structures
   1. Nesting if/switch/loops
   2. *“goto”*-like / *“spaghetti”* code (can exist even without a ***goto*** statement!)
4. Duplications or near-duplications
   1. Code duplications
   2. Member duplications
5. Too long functions/methods
6. Too many function/method parameters
7. Raw arrays
8. Writing code with the sole purpose to “silence” the compiler / Coverity
9. Preparing to call a function/method
10. Exact real number checks (using *==* or *!=* to compare floats/doubles)
11. Dead code, commented-out code, etc.
12. Relying on indirect ***#include-***s for types we directly use
13. ***using***-directives in headers (i.e. *using namespace std;*)
14. Checking for nullpointer before *delete*
15. Multiple classes per translation unit
16. Too many comments
17. Too few comments
18. **OOP / Class Design**
19. Lack of ***virtual destructors*** for base classes / interfaces
20. Base classes depending on their descendants
21. Large classes
22. “***Data classes***” – that have only data members (not to confuse with ***value classes*** like std::string)
23. Classes with only / mainly ***static methods*** in them
24. Exporting class’s internals to the user
    1. Public non-***const*** fields
    2. Protected non-***const*** fields – lesser evil but still not good
    3. Leaking unnecessary details to the user
25. ***Type checking***, no matter the form (switching on ***type*** or using ***dynamic\_cast*** etc.)
26. “For-tests only” methods in production classes
27. Accessors (getters and setters)
28. Methods that make more use of another class than their own
29. A class that changes for many different reasons
30. Fixes / changes frequently requiring changes in multiple classes
31. Using ***union***
32. Classes that just pass data around and don’t have added value
33. Primitive obsession
    1. Passing / containing set of variables/types that often go together (i.e. *foo(int x, int y)*)
    2. Using primitive type (i.e. double) for everything: lengths, frequencies, volumes, weight, angle …
34. ***new*** all over the code
35. Exposed raw collections (i.e. vector<Contact>) instead of a dedicated class
36. Circular dependencies
37. Forward declarations (***class X;***)
38. Depending on concrete classes
39. Over-engineering
40. Creating your collaborators
41. *Using.long.chains->of.member->access*
42. Class / method depending on a very small part of another class
43. **Error handling**
44. Throwing by pointer
45. Catching by value
46. Using exceptions for normal code flow
47. Using exceptions for situations that cannot be handled (i.e. programming errors)
48. Silently ignoring an error
49. Catching exceptions thrown in the same function/method
50. Checking whether “the world still works” (conditions that are both absolutely sure and outside your code)
51. Excessive checks
52. **Resource management**
53. Using raw **owning** pointers if you have *smart pointers* available
54. Managing multiple resources per class (except container classes)
55. Managing / cleaning up resources belonging to other classes
56. **Efficiency – performance, memory usage, build times**
57. Parameters/return values passed *by value*
58. Large objects/data structures in stack
59. Copying large objects / data structures
60. Unused ***#include***-s or ***using***-directives
61. Bad locality
    1. Bad data locality (i.e. **linked lists** & **sets** instead of **std::vector**)
    2. Bad processing locality (i.e. loop on 2D array by col-row instead of row-col)
62. (C++11 and above) Use of ***std::map***, ***std::set*** instead of ***std::unordered\_xxx*** counterparts
63. ***Virtual destructors*** for classes not intended to be inherited