<http://en.wikipedia.org/wiki/Code_smell>

Code smell

From Wikipedia, the free encyclopedia

In [computer programming](http://en.wikipedia.org/wiki/Computer_programming), **code smell** is any [symptom](http://en.wikipedia.org/wiki/Symptom) in the [source code](http://en.wikipedia.org/wiki/Source_code) of a [program](http://en.wikipedia.org/wiki/Computer_program) that possibly indicates a deeper problem. Code smells are usually not [bugs](http://en.wikipedia.org/wiki/Software_bug)—they are not technically incorrect and don't currently prevent the program from functioning. Instead, they indicate weaknesses in design that may be slowing down development or increasing the risk of bugs or failures in the future.

Often the deeper problem hinted by a code smell can be uncovered when the code is subjected to a short [feedback cycle](http://en.wikipedia.org/wiki/Feedback) where it is[refactored](http://en.wikipedia.org/wiki/Refactoring) in small, controlled steps, and the resulting design is examined to see if there are any further code smells that indicate the need of more refactoring. From the point of view of a programmer charged with performing refactoring, code smells are [heuristics](http://en.wikipedia.org/wiki/Heuristic_(computer_science)) to indicate when to refactor, and what specific refactoring techniques to use. Thus, a code smell is a driver for refactoring.

The term appears to have been coined by [Kent Beck](http://en.wikipedia.org/wiki/Kent_Beck) on [WardsWiki](http://en.wikipedia.org/wiki/WardsWiki" \o "WardsWiki) in the late 1990s. Usage of the term increased after it was featured in *Refactoring: Improving the Design of Existing Code*.[[1]](http://en.wikipedia.org/wiki/Code_smell#cite_note-1) *Code smell* is also a term used by [agile](http://en.wikipedia.org/wiki/Agile_software_development) programmers.[[2]](http://en.wikipedia.org/wiki/Code_smell#cite_note-In_Praise_Of_Small_Code-2)

Determining what is and is not a code smell is often a subjective judgment, and will often vary by language, developer and development methodology. There are tools, such as [Checkstyle](http://en.wikipedia.org/wiki/Checkstyle" \o "Checkstyle), [PMD](http://en.wikipedia.org/wiki/PMD_(software)) and [FindBugs](http://en.wikipedia.org/wiki/FindBugs" \o "FindBugs) for [Java](http://en.wikipedia.org/wiki/Java_(programming_language)), to automatically check for certain kinds of code smells.

|  |
| --- |
| **Contents**    [[hide](http://en.wikipedia.org/wiki/Code_smell)]   * [1 Common code smells](http://en.wikipedia.org/wiki/Code_smell#Common_code_smells) * [2 See also](http://en.wikipedia.org/wiki/Code_smell#See_also) * [3 References](http://en.wikipedia.org/wiki/Code_smell#References) * [4 External links](http://en.wikipedia.org/wiki/Code_smell#External_links) |

Common code smells [[edit](http://en.wikipedia.org/w/index.php?title=Code_smell&action=edit&section=1)]

* [*Duplicated code*](http://en.wikipedia.org/wiki/Duplicate_code): identical or very similar code exists in more than one location.
* *Long method*: a [method](http://en.wikipedia.org/wiki/Method_(computer_science)), function, or procedure that has grown too large.
* *Large class*: a [class](http://en.wikipedia.org/wiki/Class_(computer_science)) that has grown too large. See [God object](http://en.wikipedia.org/wiki/God_object).
* *Too many parameters*: a long list of parameters in a procedure or function make readability and code quality worse.
* *Feature envy*: a class that uses methods of another class excessively.
* *Inappropriate intimacy*: a class that has dependencies on implementation details of another class.
* *Refused bequest*: a class that [overrides](http://en.wikipedia.org/wiki/Method_overriding_(programming)) a method of a base class in such a way that the [contract](http://en.wikipedia.org/w/index.php?title=Contract_(software)&action=edit&redlink=1) of the [base class](http://en.wikipedia.org/wiki/Base_class) is not honored by the [derived class](http://en.wikipedia.org/wiki/Derived_class). See [Liskov substitution principle](http://en.wikipedia.org/wiki/Liskov_substitution_principle" \o "Liskov substitution principle).
* *Lazy class / Freeloader*: a class that does too little.
* *Contrived complexity*: forced usage of overly complicated [design patterns](http://en.wikipedia.org/wiki/Design_pattern_(computer_science)) where simpler design would suffice.
* *Excessively long identifiers*: in particular, the use of [naming conventions](http://en.wikipedia.org/wiki/Naming_convention_(programming)) to provide disambiguation that should be implicit in the[software architecture](http://en.wikipedia.org/wiki/Software_architecture).
* *Excessively short identifiers*: the name of a variable should reflect its function unless the function is obvious.
* *Excessive use of literals*: these should be coded as named constants, to improve readability and to avoid programming errors. Additionally, [literals](http://en.wikipedia.org/wiki/Literal_(computer_programming)) can and should be externalized into resource files/scripts where possible, to facilitate localization of software if it is intended to be deployed in different regions.
* *Ubercallback*: a [callback](http://en.wikipedia.org/wiki/Callback_(computer_programming)) that is trying to do everything
* *Complex conditionals*: branches that check lots of unrelated conditions and edge cases that don't seem to capture the meaning of a block of code.

See also [[edit](http://en.wikipedia.org/w/index.php?title=Code_smell&action=edit&section=2)]

* [Anti-pattern](http://en.wikipedia.org/wiki/Anti-pattern)
* Category [Programming principles](http://en.wikipedia.org/wiki/Category:Programming_principles)
* Category [Programming rules of thumb](http://en.wikipedia.org/wiki/Category:Programming_rules_of_thumb)
* [List of tools for static code analysis](http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)

References [[edit](http://en.wikipedia.org/w/index.php?title=Code_smell&action=edit&section=3)]

* 1. [**^**](http://en.wikipedia.org/wiki/Code_smell#cite_ref-1) [Fowler, Martin](http://en.wikipedia.org/wiki/Martin_Fowler) (1999). *Refactoring. Improving the Design of Existing Code*. Addison-Wesley. [ISBN](http://en.wikipedia.org/wiki/International_Standard_Book_Number) [0-201-48567-2](http://en.wikipedia.org/wiki/Special:BookSources/0-201-48567-2).
  2. [**^**](http://en.wikipedia.org/wiki/Code_smell#cite_ref-In_Praise_Of_Small_Code_2-0) Andrew Binstock (2011-06-27). ["In Praise Of Small Code"](http://www.informationweek.com/news/development/architecture-design/231000038). Information Week. Retrieved 2011-06-27.

External links [[edit](http://en.wikipedia.org/w/index.php?title=Code_smell&action=edit&section=4)]

* [CodeSmell at c2.com](http://c2.com/cgi/wiki?CodeSmell)
* [Taxonomy of code smells](http://www.soberit.hut.fi/mmantyla/BadCodeSmellsTaxonomy.htm)
* [Overview of many code smells](http://www.codinghorror.com/blog/2006/05/code-smells.html)