WIKIPEDIA

Code smell

In computer programming, a **code smell** is any characteristic in the source code of a program that possibly indicates a deeper problem.[1][2] Determining what is and is not a code smell is subjective, and varies by language, developer, and development methodology.

The term was popularised by Kent Beck on WardsWiki in the late 1990s. Usage of the term increased after it was featured in the book *Refactoring: Improving the Design of Existing Code* by Martin Fowler.[3] It is also a term used by agile programmers.[4]

Contents

Definition

One way to look at smells is with respect to principles and quality: "Smells are certain structures in the code that indicate violation of fundamental design principles and negatively impact design quality".[5] Code smells are usually not bugs; they are not technically incorrect and do not prevent the program from functioning. Instead, they indicate weaknesses in design that may slow down development or increase the risk of bugs or failures in the future. Bad code smells can be an indicator of factors that contribute to technical debt.[1] Robert C. Martin calls a list of code smells a "value system" for software craftsmanship.[6]

Often the deeper problem hinted at by a code smell can be uncovered when the code is subjected to a short feedback cycle, where it is refactored in small, controlled steps, and the resulting design is examined to see if there are any further code smells that in turn indicate the need for more refactoring. From the point of view of a programmer charged with performing refactoring, code smells are heuristics to indicate when to refactor, and what specific refactoring techniques to use. Thus, a code smell is a driver for refactoring.

A 2015 study[1] utilizing automated analysis for half a million source code commits and the manual examination of 9,164 commits determined to exhibit "code smells" found that:

- There exists empirical evidence for the consequences of "technical debt", but there exists only anecdotal evidence as to *how*, *when*, or *why* this occurs.
- "Common wisdom suggests that urgent maintenance activities and pressure to deliver features while prioritizing time-to-market over code quality are often the causes of such smells".

There are tools, such as Checkstyle, PMD, and FindBugs for Java source code, to automatically check for certain kinds of code smells.

Common code smells

Application-level smells:

- *Duplicated code*: identical or very similar code exists in more than one location.
- *Contrived complexity*: forced usage of overcomplicated design patterns where simpler design would suffice.
- *Shotgun surgery*: a single change needs to be applied to multiple classes at the same time.

Class-level smells:

- *Large class*: a class that has grown too large. See God object.
- *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 a method of a base class in such a way that the contract of the base class is not honored by the derived class. See Liskov substitution principle.
- *Lazy class / freeloader*: a class that does too little.
- *Excessive use of literals*: these should be coded as named constants, to improve readability and to avoid programming errors. Additionally, literals can and should be externalized into resource files/scripts, or other data stores such as databases where possible, to facilitate localization of software if it is intended to be deployed in different regions.
- *Cyclomatic complexity*: too many branches or loops; this may indicate a function needs to be broken up into smaller functions, or that it has potential for simplification.
- *Downcasting*: a type cast which breaks the abstraction model; the abstraction may have to be refactored or eliminated.[7]
- *Orphan variable or constant class*: a class that typically has a collection of constants which belong elsewhere where those constants should be owned by one of the other member classes.
- *Data clump*: Occurs when a group of variables are passed around together in various parts of the program. In general, this suggests that it would be more appropriate to formally group the different variables together into a single object, and pass around only this object instead.[8][9]

Method-level smells:

- *Too many parameters*: a long list of parameters is hard to read, and makes calling and testing the function complicated. It may indicate that the purpose of the function is ill-conceived and that the code should be refactored so responsibility is assigned in a more clean-cut way.
- *Long method*: a method, function, or procedure that has grown too large.
- *Excessively long identifiers*: in particular, the use of naming conventions to provide disambiguation that should be implicit in the software architecture.
- *Excessively short identifiers*: the name of a variable should reflect its function unless the function is obvious.
- *Excessive return of data*: a function or method that returns more than what each of its callers needs.
- *Excessively long line of code* (or God Line): A line of code which is too long, making the code difficult to read, understand, debug, refactor, or even identify possibilities of software reuse. Example:

```
new XYZ(s).doSomething(buildParam1(x), buildParam2(x), buildParam3(x), a + Math.sin(x)*Math.tan(x*y + z)).doAnythingElse().build().sendRequest();
```

See also

- Design smell
- Anti-pattern
- List of tools for static code analysis
- Software rot

References

1. Tufano, M.; Palomba, F.; Bavota, G.; Oliveto, R.; Di Penta, M.; De Lucia, A.; Poshyvanyk, D. (2015-05-01). "When and Why Your Code Starts to Smell Bad" (http://www.cs.wm.edu/~denys/pubs/ICSE'15-BadSmells-CRC.pdf) (PDF). *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*. **1**: 403–414. CiteSeerX 10.1.1.709.6783 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.709.6783). doi:10.1109/ICSE.2015.59 (https://doi.org/10.1109%2FICSE.2015.59). ISBN 978-1-4799-1934-5.
2. Fowler, Martin. "CodeSmell" (http://martinfowler.com/bliki/CodeSmell.html). *martinfowler.com/*. Retrieved 19 November 2014.
3. Fowler, Martin (1999). *Refactoring. Improving the Design of Existing Code*. Addison-Wesley. ISBN 978-0-201-48567-7.
4. Binstock, Andrew (2011-06-27). "In Praise Of Small Code" (http://www.informationweek.com/architecture/in-praise-of-small-code/d/d-id/1098422). Information Week. Retrieved 2011-06-27.
5. Suryanarayana, Girish (November 2014). *Refactoring for Software Design Smells*. Morgan Kaufmann. p. 258. ISBN 978-0128013977.
6. Martin, Robert C. (2009). "17: Smells and Heuristics". *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall. ISBN 978-0-13-235088-4.
7. Miller, Jeremy. "Downcasting is a code smell" (http://codebetter.com/jeremymiller/2006/12/26/downcasting-is-a-code-smell/). Retrieved 4 December 2014.
8. Fowler, Martin. "DataClump" (https://martinfowler.com/bliki/DataClump.html). Retrieved 2017-02-03.
9. "Design Patterns and Refactoring" (https://sourcemaking.com/refactoring/smells/data-clumps). *sourcemaking.com*. Retrieved 2017-02-04.

Further reading

- Garousi, Vahid; Küçük, Barış (2018). "Smells in software test code: A survey of knowledge in industry and academia" (http://www.sciencedirect.com/science/article/pii/S0164121217303060). *Journal of Systems and Software*. **138**: 52–81. doi:10.1016/j.jss.2017.12.013 (https://doi.org/10.1016%2Fj.jss.2017.12.013).
- Sharma, Tushar; Spinellis, Diomidis (2018). "A survey on software smells" (http://www.sciencedirect.com/science/article/pii/S0164121217303114). *Journal of Systems and Software*. **138**: 158–173. doi:10.1016/j.jss.2017.12.034 (https://doi.org/10.1016%2Fj.jss.2017.12.034).

External links

- CodeSmell at c2.com (http://c2.com/cgi/wiki?CodeSmell)
- Taxonomy of code smells (http://badcodesmellstaxonomy.mikamantyla.eu/)

- Overview of many code smells (http://www.codinghorror.com/blog/2006/05/code-smells.html)
- CodeSmell (http://martinfowler.com/bliki/CodeSmell.html)
- Boundy, David, Software cancer: the seven early warning signs (http://dl.acm.org/citation.cfm?id=156632) or here (https://www.academia.edu/2303865/Software_cancer_the_seven_early_warning_signs), ACM SIGSOFT Software Engineering Notes, Vol. 18 No. 2 (April 1993), Association for Computing Machinery, New York, NY, USA

Retrieved from "https://en.wikipedia.org/w/index.php?title=Code_smell&oldid=887827109"