Code debt:

## Common code smells

## Application-level smells:

* [*Duplicated code*](https://en.wikipedia.org/wiki/Duplicate_code)
* *Contrived complexity*: forced usage of overcomplicated [design patterns](https://en.wikipedia.org/wiki/Design_pattern_(computer_science)) where simpler design would suffice.

Class-level smells:

* *Large class*: a [class](https://en.wikipedia.org/wiki/Class_(computer_science)) that has grown too large.
* *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](https://en.wikipedia.org/wiki/Method_overriding_(programming)) a method of a base class in such a way that the [contract](https://en.wikipedia.org/wiki/Contract_(software)) of the [base class](https://en.wikipedia.org/wiki/Base_class) is not honored by the [derived class](https://en.wikipedia.org/wiki/Derived_class). See [Liskov substitution principle](https://en.wikipedia.org/wiki/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](https://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.
* [*Cyclomatic complexity*](https://en.wikipedia.org/wiki/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*](https://en.wikipedia.org/wiki/Downcasting): a type cast which breaks the abstraction model; the abstraction may have to be refactored or eliminated.
* *Orphan variable or constant class*: a [class](https://en.wikipedia.org/wiki/Class_(computer_science)) 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*](https://en.wikipedia.org/wiki/Data_Clump_(Code_Smell)): 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]](https://en.wikipedia.org/wiki/Code_smell#cite_note-8)[[9]](https://en.wikipedia.org/wiki/Code_smell#cite_note-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](https://en.wikipedia.org/wiki/Method_(computer_science)), function, or procedure that has grown too large.
* *Excessively long identifiers*: in particular, the use of [naming conventions](https://en.wikipedia.org/wiki/Naming_convention_(programming)) to provide disambiguation that should be implicit in the [software architecture](https://en.wikipedia.org/wiki/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.